Escola d'Enginyeria de Telecomunicació i Aeroespacial de Castelldefels
UNIVERSITAT POLITÈCNICA DE CATALUNYA BARCELONATECH

MASTER THESIS

TITLE: Implementation of a Random Forest Machine Learning Algorithm in the context of Gaia space mission

MASTER DEGREE: Master's degree in Aerospace Science and Technology

AUTHOR: Carles Cantero Mitjans

DIRECTOR: Santiago Torres Gil

DATE: February 8, 2018

Títol: Implementació d'un Algoritme de Bosc Aleatori en el contexte de la missió espacial Gaia

Autor: Carles Cantero Mitjans

Director: Santiago Torres Gil

Data: 8 de febrer de 2018

Resum

La missió astromètrica espacial Gaia escanejarà prop de mil milions d'estrelles una mitjana de 70 vegades cadascuna al llarg de cinc anys. Durant el temps de la missió es registraran observacions astromètriques, fotomètriques i espectroscòpiques de tot el cel fins a magnitud 20. Es a dir, Gaia serà capaç de construir un mapa tridimensional complet d'aproximadament l'1% de la nostra Galàxia emmagatzemant una gran quantitat de resultats de totes les estrelles observades amb la màxima qualitat mai assolida. Tota aquesta gran quantitat de dades astronòmiques, han de ser manejades de manera eficient. La utilització d'algoritmes d'aprenentatge automàtic i altres estratègies de classificació automàtica esdevenen essencials en un marc de dades tan gran com es el de Gaia. L'objectiu principal d'aquesta tesi de màster consisteix en preparar i provar un algoritme de classificació automàtic eficient. En aquest treball es consideraran cinc models supervisats, essent l'algoritme de Bosc Aleatori el model que presenta les millor capacitats i rendiment. Aprofitarem un simulador detallat de la població de nanes blanques, a càrrec del Grup d'Astronomia i Astrofísica del Departament de Física de la UPC. Aquest simulador ens proporcionarà una població sintètica detallada que imitarà les característiques de la població observada de nanes blanques de Gaia. Aquesta població sintètica s'utilitzarà en la fase d'aprenentatge de l'algoritme de Bosc Aleatori, per tal d'optimitzar la seva implementació a les dades observades. Una vegada provat, el nostre algorisme s'aplicarà a les dades extretes de les publicacions de dades de Gaia disponibles per tal de classificar el seu contingut en les diferents subpoblacions de la Galàxia, com ara l'halo o el disc. La precisió obtinguda en el present treball pel nostre algoritme de Bosc Aleatori (85%) representa una millora substancial amb respecte d'altres mètodes clàssics (55%).

Title : Implementación de un Algoritmo de Bosque Aleatorio en el contexto de la misión espacial Gaia

Author: Carles Cantero Mitjans

Advisor: Santiago Torres Gil

Date: February 8, 2018

Overview

Gaia space astrometry mission will scan about one billion stars an average of 70 times each over five years. During the mission time repeated astrometric, photometric and spectroscopic observations of the entire sky down to magnitude 20 will be recorded. In other words, Gaia will be able to build a complete three-dimensional map of 1 per cent of our Galaxy storing a huge amount of results from all stars observed with the highest quality ever achieved. All these large amount of astronomical data must be efficiently handled. The use of machine learning algorithms and other automatic classification strategies becomes essential in such a big data frame. The main objective of this master thesis consists to prepare and tested an efficient automatize machine learning algorithm. Five supervised models are considered in this work, becoming the Random Forest algorithm the model that present the best capabilities and performance. We will take advantage of a detailed simulator of the white dwarf population, provided by the Astronomy and Astrophysics Group of the Physics Department of the UPC. This simulator will provide us with a detailed synthetic population that will mimic the characteristic of the observed population of white dwarfs by Gaia. This synthetic population will be used in the learning stage of the Random Forest Algorithm, in order to optimize its implementation to the observed data. Once tested, our algorithm has been applied to the extracted data from available Gaia Data Releases in order to classify its content in the different subpopulations of Galaxy such as the halo or the disk. The accuracy obtained in the present work by our Random Forest algorithm (85%) represents a substantial improvement with respect to other classical methods (55%).

Título: Implementación de un Algoritmo de Bosque Aleatorio en el contexto de la misión espacial Gaia

Autor: Carles Cantero Mitjans

Director: Santiago Torres Gil

Fecha: 8 de febrero de 2018

Resumen

La misión astrométrica espacial Gaia escaneará cerca de mil millones de estrelles una media de 70 veces cada una a lo largo de cinco años. Durante el tiempo de la misión se registrarán observaciones astrométricas, fotométricas y espectroscópicas de todo el cielo hasta magnitud 20. Es decir, Gaia será capaz de construir un mapa tridimensional completo de aproximadamente el 1% de nuestra Galaxia, almacenando una gran cantidad de resultados de todas las estrellas observadas con la máxima calidad nunca antes conseguida. Toda esta gran cantidad de datos astronómicos, ha de ser manejada de manera eficiente. La utilización de algoritmos de aprendizaje automático y otras estrategias de clasificación automática devienen esenciales en un marco de datos tan grande como es de Gaia. El objetivo principal de esta tesis de máster consiste en preparar y probar un algoritmo de clasificación automático eficiente. En este trabajo se considerarán cinco modelos supervisados, siendo el algoritmo de Bosque Aleatorio el modelo que presenta las mejores capacidades y rendimiento. Aprovecharemos un simulador detallado de la población de enanas blancas, proporcionado por del Grupo de Astronomía y Astrofísica del Departamento de Física de la UPC. Este simulador nos proporcionará una población sintética detallada que imitará las características de la población observada de enanas blancas de Gaia. Esta población sintética se utilizará en la fase de aprendizaje del algoritmo de Bosque Aleatorio, para optimizar su implementación en los datos observados. Una vez probado, nuestro algoritmo se aplicará a los datos extraídos de las publicaciones de datos de Gaia disponibles para clasificar su contenido en las diferentes subpoblaciones de la Galaxia, tales como el halo o el disco. La precisión obtenida en el presente trabajo por nuestro algoritmo de Bosque Aleatorio (85%) representa una mejora sustancial con respecto a otros métodos clásicos (55%).

For my parents, my entire life
Really thanks, I love you.

CONTENTS

LIST OF FIGURES

LIST OF TABLES

ACKNOWLEDGEMENTS

At first instance, I would like to acknowledge so much my advisor Dr. Santiago Torres for their patience, and for guiding and teaching me along this period, where we have been working and discussing different ideas about the project. I also want to remark the fact that he accepted me and also trusted in me without any problem to start with this amazing project, when I needed more than ever. It has been a great pleasure to share and specially acquire so much knowledge of him and with him.

Also thanks to the complete Astrophysics team of the UPC who helped me many times contributing with several ideas and improvements. Alberto thank you very much for your fantastic guidance as well, and at the same time, your experience in the field. Georgy thanks to you for your contribution with the simulations. Really thanks Miguel too, for your accompaniment during the working period and your ideas in the meeting groups. To finish with this group, I would like to do a huge mention and reference to Enrique García-Berro who was a big referent in my career for its dedication, effort and huge personality within the astronomy field.

In the personal side, I specially want to dedicate the entire work and also the huge illusion for that to my parents. They always were, are and will be who have made possible that I am here now, where I really want to be. From here, I want to transmit all the force to my dad who is passing a bad stage. I want also to thank to my cousin Marina for its unconditional support every moment.

Finally I want to mention two very important people for me who always have been there in the adequate moment. They are my friends Axel and Irmina with which I have been able to share some opinions about this project. I am also really grateful for having an special friend to my side help me always, thank you Sergi.

To sum it up I would like to finish remarking the huge contribution of my master acquaintances who always have crazy at the same time, amazing innovative ideas to go on. This year have been one of the most beautiful and interesting periods of my life, thank you guys for everything.

INTRODUCTION

It is widely known the constant increase in higher resolution power and efficiency in all space missions; Gaia Space Mission is a good example. This fact directly implies that the more precision in our machines, the more amount of data to be treated and computed. Although, it can seem a big advantage, also signify an extra-problem to deal with. Big Data treatment refers to voluminous data sets in comparison with the traditional data processing applications, which are stored and analysed by former computers. Therefore, there is a need to design new methodologies to treat with this amount of data. In the astronomical case, a high numerous of star sample (millions or maybe billions) together with all their features and parameters should be well-stored under astronomical surveys specifically created to solve this issue.

Big astronomical surveys are a reality at the moment. Current large surveys and, in particular astronomical satellites, are the main source for discovery astronomical objects and the main source of observational data for further analysis, interpretation, and eventually achieving new scientific results. Besides, they allow astronomers to catalogue celestial objects and perform statistical analyses on them without making lengthy observations.

One of the most useful astronomical surveys nowadays is the renowned Sloan Digital Sky Survey (SDSS). It is commonly seen as a major multi-spectral imaging and spectroscopic red-shift survey. By means of using a dedicated 2.5 m wide-angle optical telescope located at Apache Point Observatory, all data releases can be performed with a certain periodicity. Data collection began in 2000, and the final imaging data release covers over 35% of the sky, with photometric observations of around five hundred million objects and spectra for more than three million objects. In order to the scientific community can have access to all these data releases, SDSS provides a dedicated web-page (see [1]).

On the other hand, Gaia space astrometry mission also pretends to make a new fully Galaxy survey with a higher accuracy and power resolution never seen so far. Gaia satellite was injected into a Lissajous orbit around the Sun-Earth Lagrange point L2. This kind of orbit allows nearly uninterrupted observations of regions in the sky pointing away from the Sun. Therefore, it will scan about one billion stars an average of 70 times each over five years. During the mission time repeated astrometric, photometric and spectroscopic observations of the entire sky down to magnitude 20 will be recorded. In other words, Gaia will be able to build a complete three-dimensional map of one per cent of our entire Galaxy. Thus, the mission concept is optimized to yield a huge amount of astrometric results of all stars observed with the highest quality. So far, scientist community have only had access to the first Gaia Data Release (DR1) in which only positions (equatorial coordinates , α and δ) and G magnitudes for all sources with acceptable formal standard errors on positions are stored. The amount of data that predictably Gaia will storage (estimates of 200 TB) limits the use of home computers instead of online website resources such as the well-known Gaia Archive website.

All these large amount of astronomical data, must be efficiently handle. The use of machine learning algorithms (MLA) and other automatic classification strategies becomes essential in such a big data frame. MLA is a specific branch within the artificial intelligence science whose main goal is furnish computers, or simply software, to be capable to learn without the need to be programmed previously. These algorithms can be classified in two groups: Supervised Learning or Unsupervised Learning. In the first one, a previous

knowledge is needed. Conversely, in unsupervised learning method, artificial intelligence do not need to have a previous experience or knowledge so as to analyse the new data. Among these machine learning techniques, one of the most promising used by data scientists is Random Forest Algorithm (RFA). Basically, it consist in a multipurpose ensemble learning method for classification and regression that construct and apply a number of decision trees over initial parameters. From this learning method, stars or other objects can be classified following different ranges of their physical parameters according to all the decision trees. Single decision trees often have high variance or high bias. Random Forests attempts to mitigate the problems of high variance and high bias by averaging to find a natural balance between the two extremes. Considering that Random Forests have few parameters to tune and can be used simply with default parameter settings, they are a simple tool to use without having a model or to produce a reasonable model fast and efficiently.

Our first objective was to get acquainted with the Gaia public domain data base, already available DR1 and forthcoming releases. The objective is to extract all the information concerning a particular type of stars for which we will focus our analysis. These type of stars are called white dwarfs. These stars represent the vast majority of end points of stellar evolution. The population of white dwarfs available by Gaia will carry a huge source of information about the formation, history and age of our Galaxy.

So far, only one Gaia data releases have been exhibited to Scientific Community for its subsequent review and study. The second data releases are going in this set in coming April 2018, expecting to increase several orders of magnitude the quantity of available data. In order to be able to extract and classify all the data from white dwarfs from all information provided in this second and subsequent Gaia data releases, we would need to have ready and tested an efficient automatize Machine Learning algorithm. A supervised Random Forest algorithm is presented to be the optimal one to achieve these goals. We will take advantage of a detailed simulator of the white dwarf population, provided by the Astronomy and Astrophysics Group of the Physics Department of the UPC. This simulator will provide us with a detailed synthetic population that will mimic the characteristic of the observed population of white dwarfs by Gaia. This synthetic population will be used in the learning stage of the Random Forest algorithm, in order to optimize its implementation to the observed data. Once tested, the algorithm will be applied to the extracted data from available Gaia Data Releases in order to classify its content in the different subpopulations of Galaxy such as the halo or the disk.

CHAPTER 1. BUILDING THE SAMPLES

Our first step will be to prepare the samples to which we will apply our Random Forest Algorithm. Basically that consists in two sample: the observed sample of white dwarfs for which we want to obtain a classification of their objects, and the simulated sample that will be used as a testbed in the learning process of our algorithm. This first chapter is exclusively dedicated to explain step by step how the white dwarf sample is extracted from the Gaia database and how we build a simulated sample according to our population synthesis code. For those readers not familiarized with the white dwarf population we present in Appendix A a brief description of their main physical properties and characteristics.

1.1.. The contribution of Gaia

Gaia will survey more than one billion stars through its 5 years mission. Gaia satellite represents a huge improvement of Hipparcos[1] mission (1989-1993) in which more than a million stars were recorded with an unprecedented precision. Gaia will able to scan all these sources with G magnitudes brighter than 21 mag, observing 10,000 times as many stars as Hipparcos mission did and measuring their position and motion with 100 times greater precision. Thus, in general terms, Gaia will be able to measure the position and velocity of more than one billion stars in the Milky Way (about 1% of the total number of stars in the Galaxy) charting the three-dimensional distribution of these stars and determining their brightness, temperature, composition and motion through space. This give us an idea of how important the contribution of Gaia will be in the astronomy field with high precise data never analysed so far.

1.1.1.. Data release scenario

Given the enormous amount of data that Gaia is expected to generate, different releases are programmed through the lifetime of the mission. Gaia data will be shared to the scientific community within various packages in which there will be astrometric and photometeric parameters of the total data. The Gaia team, together with the validation of Data Processing and Analysis Consortium (DPAC), has already published the first Gaia data release (DR1)[2] at September, 2016.

In Gaia DR1, it has been scanned a total number of 1142679769 sources. All these stars are previously identified in different catalogues such as Hipparcos or Tycho-2 (see more information about catalogues in section 1.2). The astrophysical information provided by DR1 is the following:

- Position in equatorial coordinates (α, δ) for all sources with an acceptable formal standard error included. VizieR
- G magnitude for all sources.

[1] http://sci.esa.int/hipparcos/

[2] https://www.cosmos.esa.int/web/gaia/dr1

- Parallaxes and proper motions for stars in common between the Tycho-2 catalogue and Gaia.

Although, some specific features were also included:

- Photometric data of selected RR Lyrae and Cepheid variable stars.
- Positions in equatorial coordinates (α, δ) and G magnitudes for 2152 quasars.

The second data release (DR2) is expected to be publicly available in April, 2018. In this new release, the data will be more complete adding new parameters of interest:

- Parallaxes of all sources will be added to the complete list.
- G and integrated G_{BP} and G_{RP} photometric fluxes and magnitudes for all sources.
- Median radial velocities for sources brighter than $\mathrm{G_{RVS}} = 12$ mag.
- Epoch astrometry for a pre-selected list of more than 10 000 asteroids.
- Effective temperature and the line-of-sight extinction will be provided, based on the above photometric data, for stars brighter than $G = 17$ mag.

However, DR3 and DR4 (the last one) will be expected at middle to late 2020 and to the end 2022, respectively. In these releases much more information and features will be stored and exposed to the scientific community, such as including a list of exoplanets and many other magnitudes such as B and R. In [3], much more information about Gaia data releases is found as well as other specific news about the mission.

1.1.2.. Gaia archive

Among all these huge amount of data from Gaia, white dwarfs will be listed with their particular variables and features. The main problem hereafter is to be able to extract them efficiently since it is not an easy task to treat with 1 billion of stars from a computational point of view. As commented in the introduction, Gaia archive (see [2]) represents, in the present moment, the principal Gaia database where astronomers can have access to the DR1 among other cross-matched tables.

Through the website of Gaia DR1, users can manipulate the information in order to present them in different plots and histograms. We need to remark here that users have access to the data by means of ADQL form interface. ADQL is a SQL-like language which includes astronomical facilities to query a database.

1.1.3.. The Initial Gaia Source List

The Initial Gaia Source List (IGSL) was commissioned by the DPAC in 2006. It is a fully combination of the best optical astrometric and photometric information on celestial objects known so far. In other words, IGSL symbolizes a snapshot of the sky such as we

know it previously to Gaia observations. This source list was made for several specific reasons, although the main one was to arrange for a complete unified list of the total celestial sources that Gaia would be able to scan during its life mission.

IGSL was updated in 2007 (IGSL1) and 2010 (IGSL2). Afterwards, DPAC decided to reduce the size of IGSL2 by removing objects fainter than Gaia limit (basically $G < 21$ mag). A smaller, cleaner and operational list was reached with that last modification (forming a new named IGSL3, hereafter simply IGSL) whose data would stay frozen so far.

Currently, the IGSL is made up of 1222598530 entries and is composed essentially for two tables; *IgslSource* in which the main data with positional, photometric, and classification information is listed; *SourceCatalogIDs*, a list of the identifiers in the various input catalogues, as well as the Hipparcos catalogue on request from the Gaia team.

In table 1 of [6] the number of stars corresponding to each catalogue is listed. Besides, much more information about the features of IGLS and its production can also be found.

1.2.. Extracting the observed white dwarf sample

Once explained the Gaia database framework, our objective is to extract a complete list of all white dwarfs that Gaia could observe. Essentially, we will take advantage of the last version of the IGSL to extract all white dwarfs from there. In what follows, the software used to manipulate all these data is specified.

1.2.1.. TOPCAT, STILTS and VizieR

TOPCAT and STILTS (see [4] and [5] respectively) are an astronomical software used to manage easily the big amount of data from current stars catalogues. TOPCAT is an free interactive graphical viewer and editor for dataset with a tabular format. It provides a great variety of facilities for manipulation and analysis of source catalogues and other kind of tables. The program also includes a number of different astronomically important formats, such as FITS or VOTable, thus becoming a basic standard astronomy tool.

Similarly, STILITS is a set of command-line tools facilitating a number of processing tasks such format conversion, cross-matching tables, statistical calculations, data validation and plotting, among other possibilities. These two programs are normally used together depending on the requirements and goals.

TOPCAT and STILTS are automatically associated with which provides access to the most complete library of published astronomical catalogues and data tables available on line organized in a self-documented database. Query tools allow the user to select relevant data tables and to extract and to format records matching a previous given criteria. Currently, 16794 catalogues are available. For more information visit [7].

Through the *Table Access Protocol (TAP) Query*, we have access to all catalogues on VizieR that can be selected and manipulated. In particular, there is the IGSL that we look for. Different features of the IGSL are shown in Fig. 1.1, as well as a small description of the table and its content.

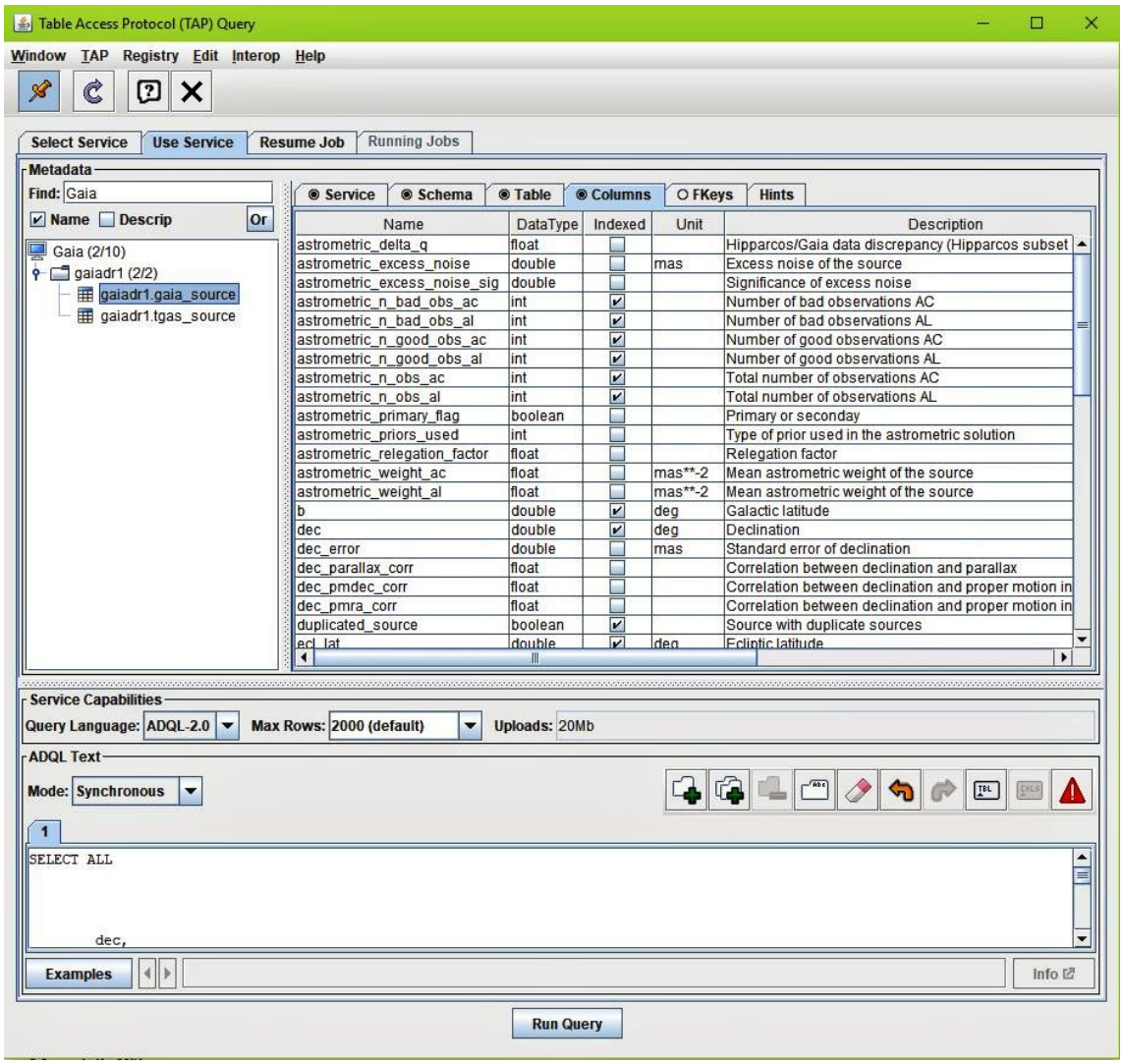

Figure 1.1: TAP query screenshot for IGSL in TOPCAT.

An specific SQL script (it can be found in Appendix B) has been made in order to extract all parameters of interest. The complete list contains:

- Equatorial coordinates (α, δ) and their corresponding errors ($\sigma_\alpha, \sigma_\delta$).
- Absolute magnitudes: B, R and G.
- Proper motions in equatorial coordinates: μ_α, μ_δ
- Total proper motion (μ) is computed as:

$$\mu = \sqrt{(\mu_\delta \cdot \cos\delta)^2 + \mu_\alpha^2} \tag{1.1}$$

The maximum number of stars that a typical home computer can extract is around 600000 over the 1222598530 total entries. It represents a too small sample comparing to the complete IGSL. Computationally, the computer cannot work efficiently with more than this number. From a statistical point of view, we are losing a lot of information about the white

dwarf sample that we want to study. Thus, it becomes a crucial problem in order to extract a significant white dwarf sample.

In consequence, instead of downloading the IGSL using the TAP Query provided by TOPCAT, we used directly the Gaia archive since a biggest number of stars was reached, specifically around one million sources. This fact give us a major probability to find a big range of white dwarfs even though one million stars are a small sample yet.

1.2.2.. Observed sample selection criteria

Next we face the problem of how to select white dwarf stars among the several type of stars that populate our Galaxy. This problem has usually been solve in the literature by means of a reduced proper motion diagram (RPMD) [8]. The reduced proper motion, H, is defined as:

$$H = m + 5 \cdot log(\mu) + 5 = M + 5 \cdot log(V_{\mathrm{TAN}}) \tag{1.2}$$

where m is the apparent magnitude of the star and μ its proper motion. Analogously we can define it through M, the absolute magnitude and V_{TAN} the tangential velocity of the object. The reduced proper motion represents an absolute magnitude of the star, that is, in independent of their location with respect to the sun. In Figure 1.2 we show the RPMD for one million sources from IGSL; in the X-axis is plotted the colour $(B-R)$ and in the Y-axis the reduced proper motion, H.

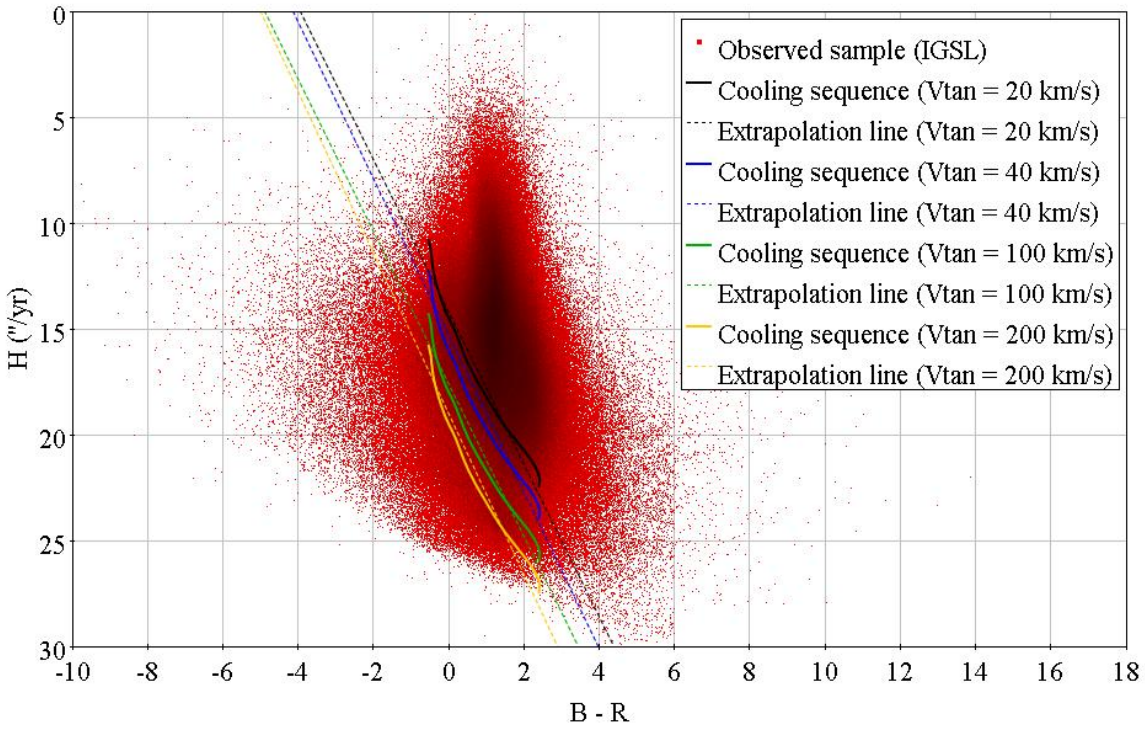

Figure 1.2: Reduced proper motion diagram for one million sources from IGSL. Also plotted different white dwarf cooling sequences for different tangential velocities and their corresponding extrapolated selection lines.

Using TOPCAT, the RPMD of our million stars can be plotted and analysed. As clearly seen in Fig. 1.2 a huge spot of stars locate in a range of H between 5 and 25"yr^{-1},

and $B-R$ from -6 to 8. Specific patterns are not easily identified in this figure, hence we cannot extract much more information since there exist a big mixture of the several galactic populations: main-sequence stars, red-giants, white dwarfs, subdwarfs stars, among other examples.

The cooling process of white dwarfs can be represented by a theoretic curve in the RPMD. In Fig. 1.2 are represented a cooling sequence for a typical $0.6\mathrm{M}_{\odot}$ white dwarf with hydrogen atmosphere for several tangential velocities. These cooling tracks indicates the region of the RPMD where white dwarfs locate. For each of the four cooling sequences curves plotted in Fig. 1.2 we also shown the corresponding extrapolated line. These lines will serve as a selection cut in order to identify the white dwarf population. The typical tangential velocity used for white dwarfs of the Galactic disk is 20 km/s. This value represents a compromise between the number of objects that we select as white dwarfs and the number of possible contaminants. That is, as shown in Fig. 1.2, as the tangential velocity increases a smaller number of white dwarfs will be selected, but on the contrary, for lower values of the tangential velocity the contamination from other sources rather than white dwarf increases. Consequently, for the main purpose of this project we adopted the standard value of 20 km/s for the tangential velocity (1.3).

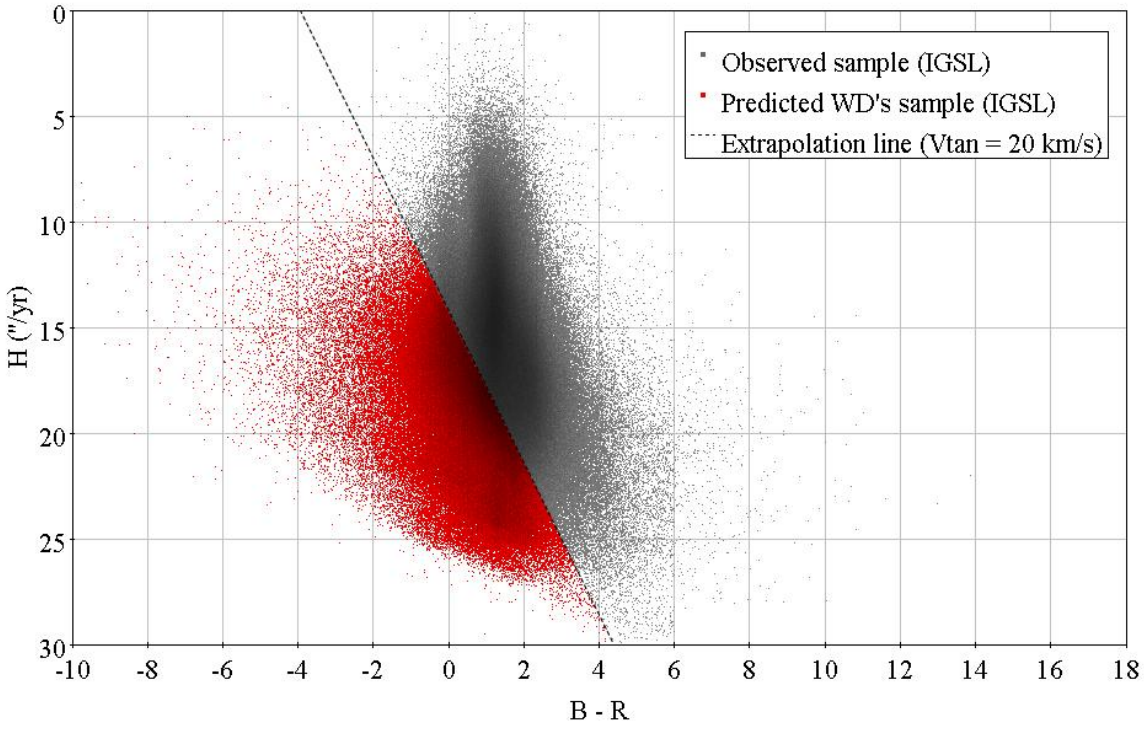

Figure 1.3: Reduced proper motion diagram showing the predicted white dwarf population obtained from the selection cut line.

Consequently, a new script on ADQL is made in order to implement the selection cut directly to Gaia Archive. As previously stated, this selection cut consists in an extrapolated line obtained from the cooling sequence for a tangential velocity of 20 Km/s. We will consider that an object is a white dwarfs if it lays below the selection cut line, that is, if it verifies the following inequality:

$$a(B-R)+b-H<0 \tag{1.3}$$

where a represents the slope of the extrapolated line and it is equal to 4.1"yr^{-1}mag, b is the H-intercept and it is equal to 15.5"yr^{-1}. The results obtained are plotted in Figure 1.3, where those objects below the selection cut line, predicted as white dwarfs, are shown as red dots.

For the sample already obtained, TOPCAT, as well as STILTS allows us to derive different statistics about the data, as can be and the range of each parameter (minimum, maximum) and the mean value. We recall here that we are interested in extract the ranges for equatorial coordinates, photometric magnitudes such as B,R and G, and proper motions. All these parameters can be seen in the table 1.1. It is also worth to saying that, in principle, we could obtain and store more data and parameters from the observed sample. However, this is not our objective. We need these optimal and feasible ranges of values in order to create a new sample of simulated white dwarfs. Eventually, this new simulated sample, will be used by our machine learning program as a testbed.

Variable	Mean	Minimum	Maximum
δ (^{o})	65.471	-36.989	88.976
α (^{o})	219.103	0.0191	359.966
B (mag)	20.726	1.758	23.961
R (mag)	19.744	17.685	29.71
G (mag)	19.757	4.723	20.998
μ_δ ("yr^{-1})	327.889	-9840.211	9827.322
μ_α ("yr^{-1})	1017.967	-9839.241	9834.512
μ ("yr^{-1})	6.7889	0.298	9.973
H ("yr^{-1})	29.708	8.546	32.313

Table 1.1: White dwarf parameter ranges from the observed sample.

1.3.. The white dwarf simulated sample

We will take advantage of the Monte Carlo simulator of the white dwarf population of the Galaxy provided by the Astronomy and Astrophysics Group of the Physic Department of the UPC. This code has been successfully applied for long time in several studies of the white dwarf population of the Galactic disk, their kinematic properties as well as in globular and open cluster of the Galaxy, among other examples. (For a description of the code see [9] and [10]). It incorporates detailed models of the galactic components as well as the most recent treatment in the evolution and cooling of white dwarfs. The synthetic sample thus obtained reproduces the characteristics of the observed sample in the range of the RPM diagram previously discussed. The basis inputs of the simulated population are as follows: for the thick disk population we used a disk age of 9.0 Gyr, a constant star formation rate, a solar constant metallicity model and a typical scale height of 250 pc; for the thin disk a burst of 1 Gyr of formation that occurred 10 Gyr ago was adopted as well as a scale height of 1000 pc and solar and subsolar metallicity model; for the halo 1 Gyr burst located 13 Gyr ago for a spherical mass distribution model with subsolar metallicity. The sample thus generated contains 165000 white dwarfs distributed in three subpopulations of the Galaxy: halo, thick and thin stars. The physical parameters provided by this

new sample are: mass ($M_\odot$), equatorial coordinates, $\alpha(^o)$ and $\delta(^o)$, J (mag), B (mag), b (mag), R (mag), r (mag), V (mag), I (mag), G (mag) and proper motions, $\mu_\alpha\,(\text{”yr}^{-1})$ and $\mu_\delta\,(\text{”yr}^{-1})$. Besides, each star is identified to which population belongs through a label. All these data has been transferred to the TOPCAT software for a better management. Thus a thoroughly analysis of the simulated data is performed in order to be compared with the observed data previously obtained.

In Fig.1.4 we represented the RPMD of the simulated sample. The white dwarf sample follows a drop shape ranging H between 10 up to 50”yr^{-1} and $B-R$ color from -0.5 up to 8. Although most stars are located between $10 < H < 30\text{”yr}^{-1}$. Even that our simulated sample extend far beyond $H > 30\text{”yr}^{-1}$, these objects are, with the current telescope resolution, quite hard to observe. It can be verified just looking the paucity of objects as we approach to this limit in Fig. 1.2. Even Gaia would be incapable to reach this limit. In view of this, an extra-limit is considered. From herein, we are going to consider a maximum value of the reduced proper motion $H = 30\,(\text{”yr}^{-1})$. Thus, it will be our limit in our machine learning program as well. Finally, for comparative purposes, we also shown in Fig.1.4 the extrapolated line of the cooling sequences for different tangential velocities as previously done in Fig.1.2.

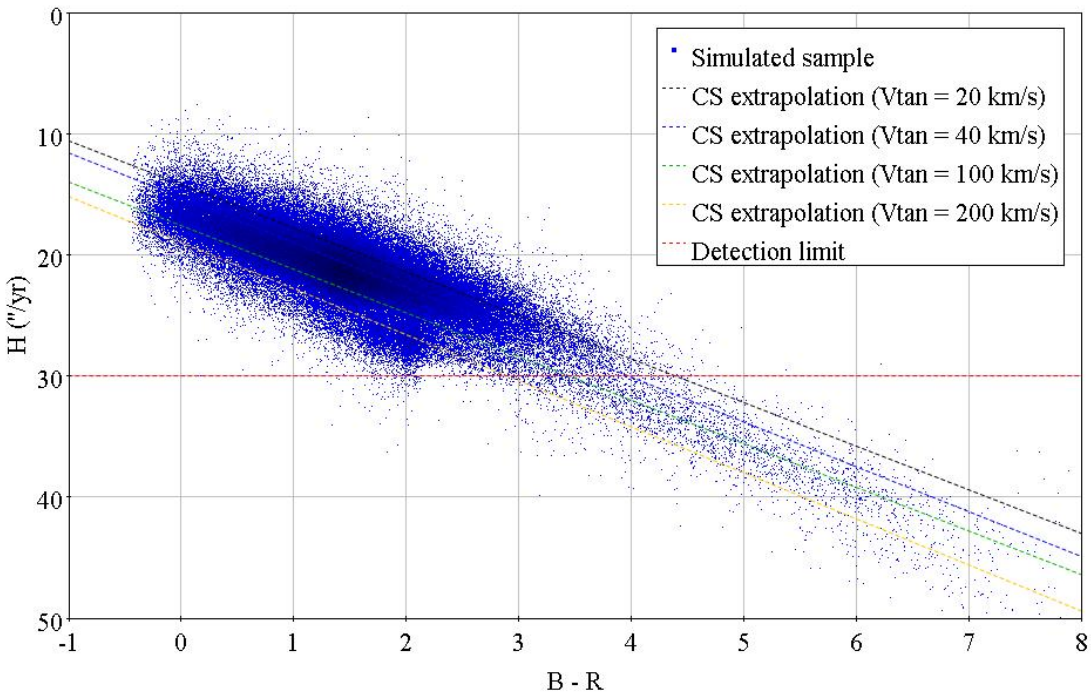

Figure 1.4: Reduced proper motion diagram of the white dwarf simulated sample with their corresponding selection cut lines for different tangential velocities.

1.3.1.. Photometric and astrometric errors

The simulated data used so far has only implemented the contribution of the Galactic extinction error. However, it is very important to keep in mind that the data from Gaia has an associated error. This error need to be mimic by our simulations. We will take advantage of one the Gaia teams, which elaborated an exhaustive document (see [11]) explaining the astrometric, photometric and spectroscopic errors on Gaia data. Therefore,

we are able to associate a photometric and astrometric error in the simulated data using the error associated to Gaia.

Both astrometric and photometric errors has smartly been parametrized by means of z in order to become calculations easier. The standard-error calculation includes all known instrumental effects, including stray-light as measured during the in-orbit commissioning phase. A detailed description of the errors involved in Gaia can be found in [12], while only a brief description is presented as follows:

- The wavelength coverage of the astrometric instrument, defining the white-light G band, is 330-1050 nm. A simple performance model which reproduces the single-field-of-view-transit G-band photometric standard errors is showed in equation 1.4.

$$\sigma_G = 10^{-3} \cdot \sqrt{0.04895z^2 + 1.8633z + 0.0001985} \tag{1.4}$$

  where $z = MAX[10^{0.4(12-15)}, 10^{0.4(G-15)}]$

- The spectral dispersion of the photometric instrument is a function of the wavelength and varies in B from 3 to 27 nm pixel^{-1} covering the wavelength range 330-680 nm. In R, the wavelength range is 640-1050 nm with a spectral dispersion of 7 to 15 nm pixel^{-1}. Equations 1.5 and 1.6 have a,b and c parameters in which the $(V-I)$ dependence is found.

$$\sigma_B = 10^{-3} \cdot \sqrt{10^{aB}z^2 + 10^{bB}z + 10^{cB}} \tag{1.5}$$

$$\sigma_R = 10^{-3} \cdot \sqrt{10^{aR}z^2 + 10^{bR}z + 10^{cR}} \tag{1.6}$$

  where $z = MAX[10^{0.4(11-15)}, 10^{0.4(G-15)}]$

- A simple performance model, including a $V-I$ colour term representing the widening of the point spread function at longer wavelengths, which reproduces the end-of-mission parallax-standard-error estimates in equation 1.7.

$$\sigma_\pi(\mu as) = \sqrt{32.732z^2 + 638.766 - 1.631} \cdot [0.986 + 0.014(V-I)] \tag{1.7}$$

  where $z = MAX[10^{0.4(12.09-15)}, 10^{0.4(G-15)}]$

- Proper motion errors have also been included in the simulations. According to the model, the error on proper motion is proportional to σ_π as equation 1.8.

$$\sigma_\mu = 0.526 \cdot \sigma_\pi \tag{1.8}$$

Using TOPCAT, we have been able to incorporate all errors to the simulated data. The results are shown in Fig.1.5, where we superimposed in the RPM diagram the simulated sample with and without photometric errors, blue and gray dots, respectively. Also plotted in Fig.1.5 are the average error bar as a function of the reduced proper motion H up to 28 magnitude. Error bars for stars with $H > 28,$"yr^{-1} are not plotted since that they are extremely large. This fact corroborate our previous assumption that as reaching the limit of

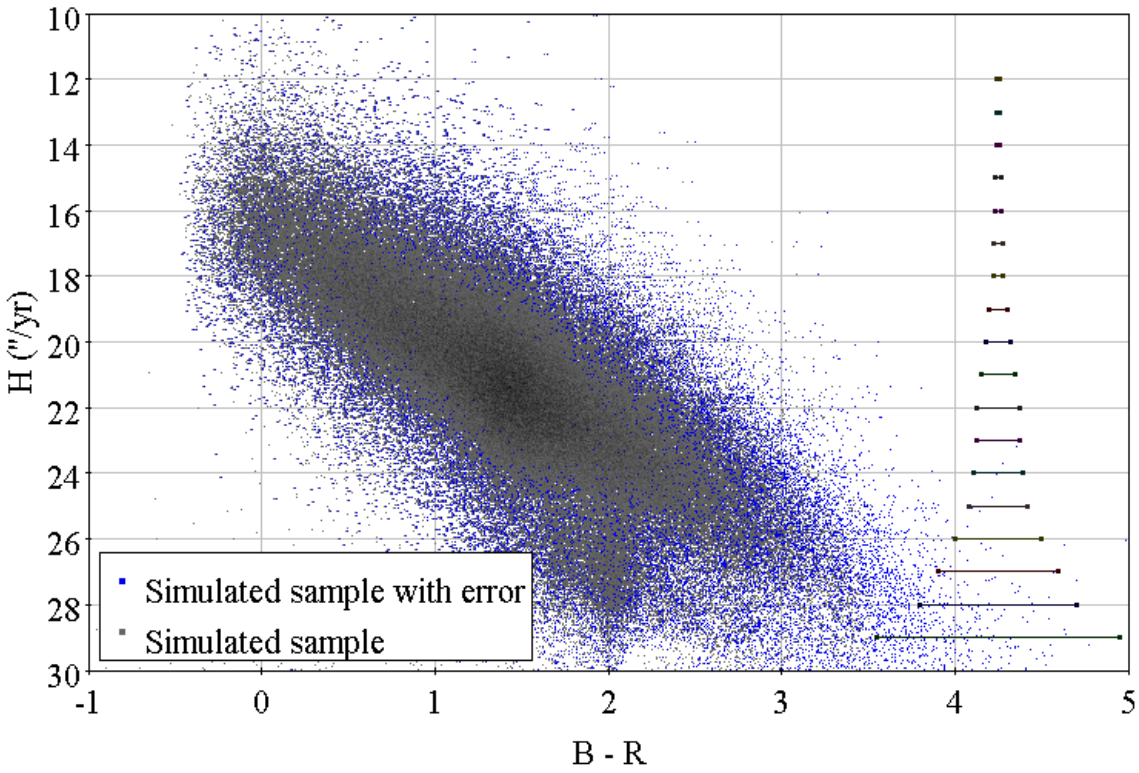

Figure 1.5: RPM diagram of the simulated sample with and without photometric errors, blue and gray dots, respectively.

$H = 30,$"yr^{-1} the accuracy of the observations significantly decreases and, consequently photometric errors become larger.

In view of Fig.1.5, we can stated that the net result of including photometric errors is an spread in the RPD diagram locus where we can find the white dwarf population as well as this spread is more prominent for cooler objects, those with high reduced proper motion, that eventually will correspond to the thick disk and halo white dwarfs. Finally, it is also worth to mention that other errors have been also considered in our simulation, such as the parallax or distance error.

1.4.. Comparing simulated and observed sample

Finally, in the last section of this chapter, we will compare both observed (IGSL) and simulated sample, in order to extract some conclusions about the selection process performed so far. It is important to recall that the simulated sample is one who brings the information to which subpopulation belongs each of the white dwarfs of the sample. This information will be crucial for starting to program our Random Forest algorithm. Given that, once the algorithm has been trained it will be applied to the original observed sample extracted from IGSL. For that reason, it is of huge interest to compare, in a first stage, the simulated and the observed samples.

The RPM diagram for the simulated and observed samples, blue and red dots, respectively, is shown in Figure1.6. The first interesting thing of Fig. 1.6 is that almost all simulated data

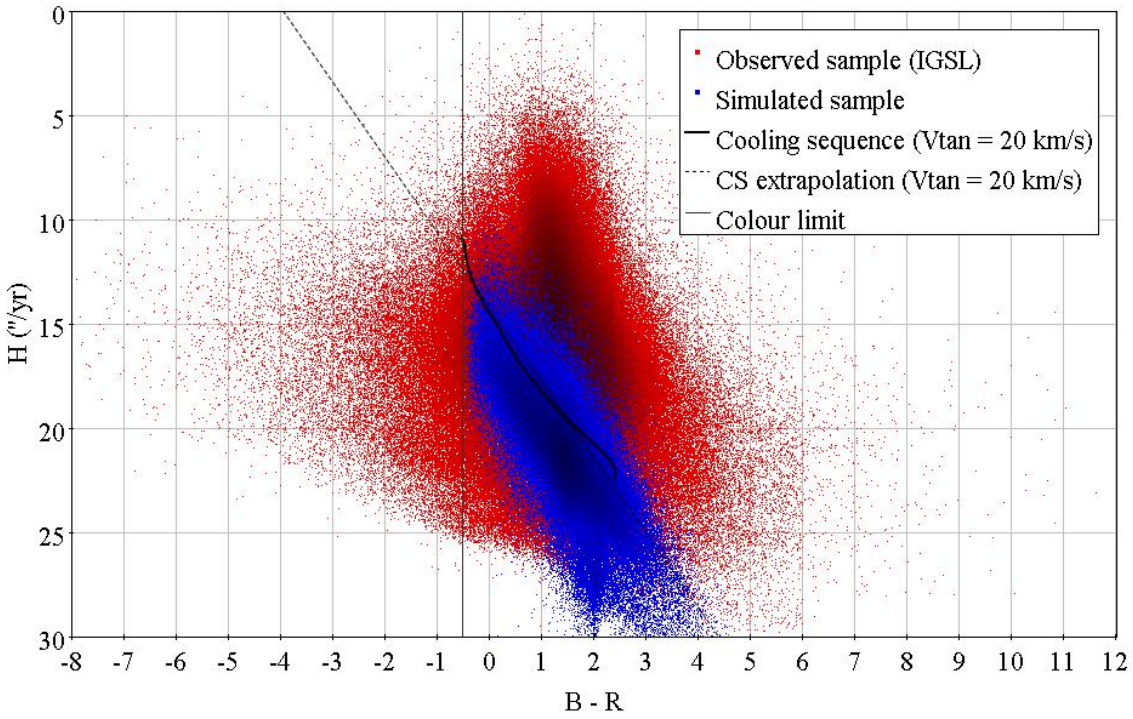

Figure 1.6: RPM diagram for the simulated and observed samples, blue and red dots, respectively.

is, as it should be, inside the observed data. Secondly, we can now see that our selection cut (represented in Fig.1.6 by a line) eliminates only a small portion of white dwarfs. This is the price that we must pay in order to ensure a low contamination of main-sequence or other type of stars. The last thing that we should point out is that the observed sample present too many objects with cooler colours (more negative values of $(B-R)$). These objects are probably not white dwarfs and consequently they should be discarded from our selected sample of observed stars.

CHAPTER 2. MACHINE LEARNING

In our current technological world, the convergence of computing and communication has produced a society that feeds on information. However most of the information is in its raw form: datasets. If data is characterized as recorded facts, then information can be defined as the set of patterns, or expectations, that underlie this data. The main problem of the information is it is stored in lots of enormous datasets. As an example, Gaia will download during its nominal lifetime a total amount of 60 TB, reaching nearly 200 TB of uncompressed data on the ground. In most of the cases the only way to treat with such a huge information is using computational new algorithms.

Therefore, data mining is the extraction of implicit, previously unknown, and potentially useful information from our data. The idea is to build computer programs that sift through databases automatically, seeking regularities or patterns. By this way, strong patterns will generalize to make accurate predictions on future data.

Machine learning is an important branch within the artificial intelligence in which the technical basis of data mining is provided. It is extremely useful to extract and analyse from the raw data in databases. The process is one of abstraction: taking the data and inferring whatever structure underlie it. This chapter exhaustively describes those techniques and methodologies used to classify whatever data in order to obtain the desired classification or regression, in our case, of the white dwarf sample.

Specific libraries of Scikit-learn combined with the IDE Jetbrains Pycharm under 3.0 Python version have been used in the implementation of the algorithms in this work. Scikit-learn is a simple and efficient tools for data mining and data analysis from where different libraries of Machine learning can be selected in order to implemented the desired method.

2.1.. Flavors of machine learning

In the machine learning framework, *supervised learning*, *unsupervised learning* and *reinforcement learning* methods are defined according to their functionalities and objectives. The choice of one or another method is based on the specific conditions of the data to analyze. First of all we need to elucidate which is the preferable technique for our sample data.

2.1.1.. Supervised learning

Supervised learning algorithms make predictions based on a set of *a priori* known examples. In other words, given some known input variables $\{x\}$ and a known output variable y, you use the algorithm *to learn* the mapping function from a new input $\{x'\}$ to its new output ($y' = f(x')$). For instance, simulated sample of white dwarfs can be used to classify other set of stars. Each example used for training is labeled with the value of interest — in our case the subpopulation of the Galaxy to which belongs the star. A supervised learning algorithm looks for patterns in those value labels. It can use any piece of information that might be relevant looking for different types of patterns. After the algorithm has found the best pattern it can make predictions for unlabeled testing data.

According to its specific characteristics supervised learning algorithm can be classified in three types: classification, regression, and anomaly detection.

- **Classification**: When the data are being used to predict a category, supervised learning is also called classification. This is the case when assigning an image as a picture of either a 'cat' or a 'dog'. When there are only two choices, it's called two-class or binomial classification. When there are more categories, such as stars samples classification in different populations, this problem is known as multi-class classification. There are several algorithms following within that group, but probably the most used one are *Regression*, *Decision Tree*, *Random Forest*, *KNN* and *Logistic Regression*.
- **Regression**: When a value is being predicted, supervised learning is called regression. Examples of Unsupervised Learning are *Apriori algorithm*, *K-means*, among others.
- **Anomaly detection**: Sometimes the goal is to identify data points that are simply unusual. In fraud detection, for example, any highly unusual credit card spending patterns are suspect. The possible variations are so numerous and the training examples so few, that it's not feasible to learn what fraudulent activity looks like. The approach that anomaly detection takes is to simply learn what normal activity looks like and identify anything that is significantly different. Example of Reinforcement Learning are *Markov Decision Process*.

2.1.2.. Unsupervised learning

In unsupervised learning, data points have no labels associated with them. That is, when we have only an input data $\{x\}$ and no corresponding output variables $\{y\}$. The goal of an unsupervised learning algorithm is to organize the data in some way or to describe its structure. This can mean grouping it into clusters or finding different ways of looking at complex data so that it appears simpler or more organized. Analogously to the supervised learning method, unsupervised can also be classified in some groups according to their functionality: clustering, anomaly detection and neural networks.

- **Clustering**: in a clustering problem you want to discover the inherent groupings in the data. Clusters are groups of datasets very similar in terms of their variables and behavior. It is a main task of exploratory data mining, and a common technique for statistical data analysis, used in many fields, including machine learning, pattern recognition, image analysis, information retrieval, bioinformatics, data compression, and computer graphics. Typically, the basics of clusters include groups with small distances among the cluster members, dense areas of the data space, and large distance between different clusters. Thus, clustering can be formulated as a multi-objective optimization problem. The appropriate clustering algorithm and parameter settings depend on the individual data set and intended use of the results. Examples of clustering methods are, the previously refered *K-means*, *Hierarchical clustering* and some *mixed models*.
- **Dimensionally reduction**: The main idea of principal component analysis is to reduce the dimensionality of a data set consisting of many variables correlated

with each other, either heavily or lightly, while retaining the variation present in the dataset, up to the maximum extent. There are some methods to do it: Principal Component Analysis (PCA), Linear Discriminant Analysis (LDA) and Generalized Discriminant Analysis (GDA), among others.

2.2.. Selecting the algorithm

As previously seen, Machine Learning presents several ways to treat big amount of data. An schematic way to present these different options is shown in Figure 2.1. Recall that the selection of one or another Machine Learning algorithm depends on the final purpose and the type of data and information about this data that we have. In our case, we are interested only for classification methods since we want to build a new software algorithm for classifying white dwarfs starts into the different Galaxy components: thin, thick and halo. Taking advantage of our simulated sample in which each star have its corresponding label (0: halo, 1: thick, 2: thin) the supervised learning algorithm will be trained in order to achieve a new model. Then, this new trained model will be able to be applied to other white dwarf samples, for instance Gaia DR2, finding the desired classification of the data.

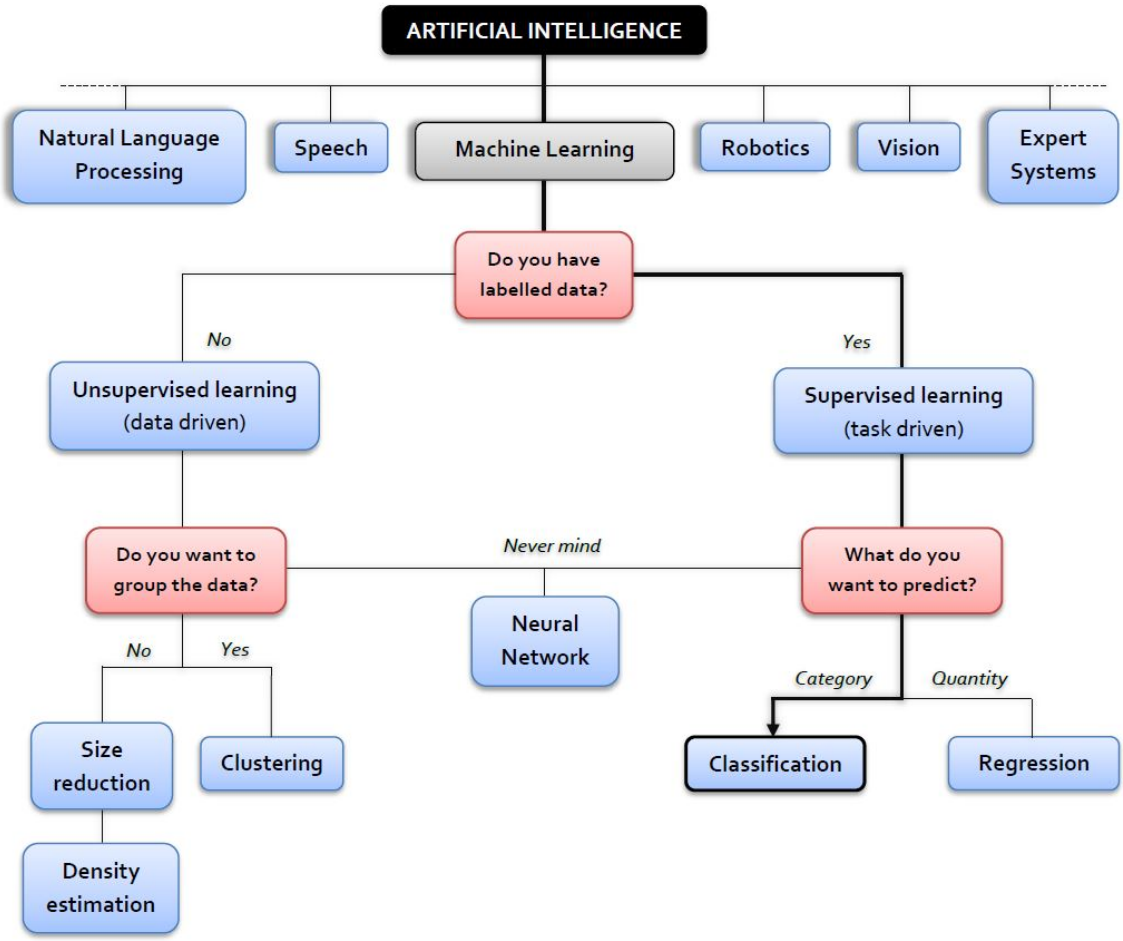

Figure 2.1: General mind map of machine learning. Black path-line indicates our choosing method to treat with our white dwarf sample.

Our selected method is indicated by a black path-line in the mind map of Fig. 2.1. In this work, we have decided to compare the predictions obtained by the different supervised methods in order to seek the best choice for our specific data. In what follows a brief description of these method is presented.

2.2.1.. Decision Tree algorithm

Decision Trees serve both as a classification and regression technique. The goal is to create a model that predicts the value of a target variable by learning simple decision rules inferred from the data features.

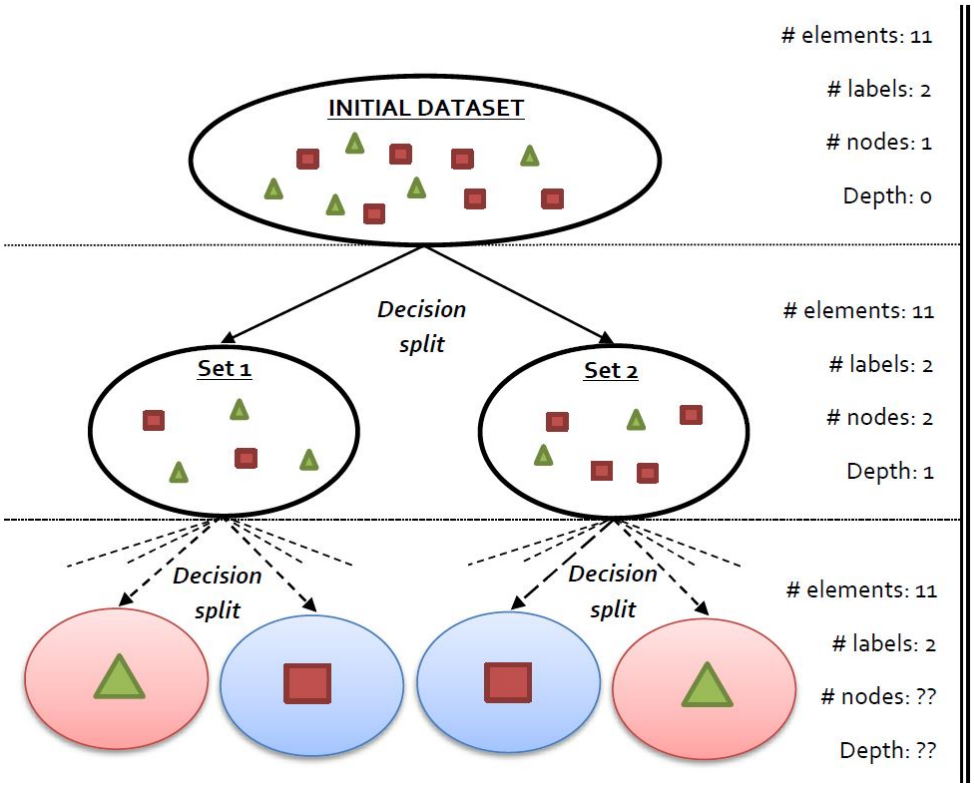

Figure 2.2: Simple example about the configuration of a decision tree result over a specific dataset.

The algorithm generates a flowchart-like structure in which we can find various elements:

- **Nodes**: They are defined as those moment in which a decision has to be made among several possible ones. Therefore, there will be more possible endings as the number of nodes increases.
- **Arrows**: They represent the union between nodes and they also are seen as every action done during the classification process.
- **Depth**: It represents how many decision splits are in the tree, or equivalently, the number of lines/flats that there are.
- **Labels**: They give a name to every action.

In Figure 2.2 a two depth level decision tree is represented with their number of nodes. Initially, there are eleven elements that are distributed as depth increases. Red squares and green triangles symbolize the labels in the data. The algorithm starts by evaluating the feature values of the elements inside the initial dataset. Based on their values, elements are put in *Set 1* or *Set 2*. In this example, after the splitting, the state seems tidier, most of the green triangles have been put in *Set 1* while a majority of red squares are in *Set 2*. Hence, decision trees order the dataset by looking at the values of the feature vector associated with each data point. Based on the values of each feature, decisions are made that eventually leads to a leaf and an answer. But the question that arises is how the algorithm is able to configure all questions in each node in order to obtain the best benefit to classify the data?

Decision Tree algorithm works in the following way: let us imagine a simplified scenario where a set of N items fall into two categories, n of them labeled as *Label 1* and $m = N - n$ labeled as *Label 2*. We introduce the ratios of the respective categories as $p = \frac{n}{N}$ and $q = \frac{m}{N} = 1 - p$. The standard Shannon entropy of our set is given by the following equation:

$$E = -\sum_i p_i \cdot log_2(p_i) = -p_i \cdot log_2(p_i) - q_i \cdot log_2(q_i) \tag{2.1}$$

In Decision Trees, at each branching, the input set is split in two (as the example of Fig. 2.2). A weighted sum of entropies is computed (weighted by the size of the two subsets):

$$E_{split} = \frac{N_1}{N} E_1 + \frac{N_2}{N} E_2 \tag{2.2}$$

where N_1 and N_2 are the number of items of each sets after the split and E_1 and E_2 are their respective entropy. The idea is at each step (each branching) to decrease the entropy. So this quantity is computed before the cut and after the cut. If entropy decreases, the split is validated and the algorithm proceeds to the next step, otherwise, it tries to split with another feature or stop this branch. Before and after the decision, the sets are different and have different sizes. Eventually, the algorithm will find the branching that minimize the entropy and consequently classified the data in the best possible way.

2.2.2.. K-Nearest Neighbours algorithm

K-Nearest Neighbours (KNN hereafter) can also be used for classification and regression purposes. In the classification setting, the KNN algorithm essentially pretends to compare the distance between one point in the data with the rest around it. This distance normally is defined as the Euclidean distance, even though other distance definitions can be more suitable for a given data, such as Chebyshev distance.

The most important parameter in this algorithm is K defined as a positive number. Basically, it represents the maximum number of neighbours around the point that we will classify. In other words, given a positive integer K, an unseen observation x and a similarity metric d, KNN classifier performs the following three steps:

1. Firstly, it computes the distance d between x (the point that we want to classify) and each training observation. The K points in the training data that are the closest to x define the set A.

2. Secondly, it is estimated the conditional probability for each class, that is, the fraction of points in A which belong to a given labeled class.

3. Finally, our input x is assigned to the class with the largest probability.

For instance, in the Fig. 2.3 we have our point x to be classified between blue or orange labels. The algorithm have pre-established a K equal to 3. This means that it will consider only those nearest three points around x in order to know to which class x belongs to. Then, we can see that x have two blues and only one orange points as neighbors. Thus the algorithm will predict that x is going to be blue.

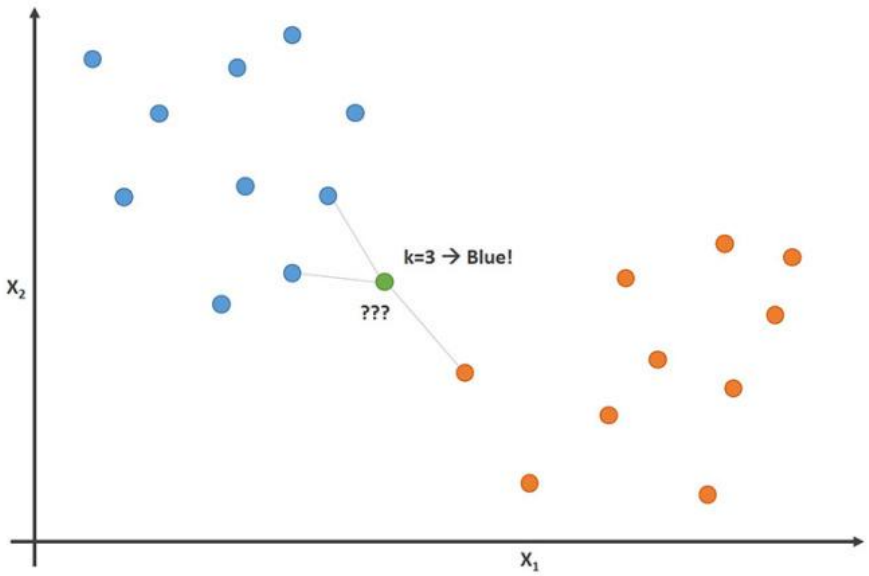

Figure 2.3: Example of how KNN algorithm works showing the K-contribution.

Finally, it is worth to saying that the proper choice of K depends on the specific characteristics of the dataset to be classified. There is not a general rule to pick up a particular value for K. After a trial an error test the best K value is selected.

2.2.3.. Ensemble methods

Ensemble methods are meta-algorithms that combine several machine learning techniques into one predictive model to produce improved results, decrease variance (bagging), bias (boosting), or improve predictions (stacking). Consequently, ensemble methods usually produces more accurate solutions than a single model would.

Essentially, two families are usually distinguished within ensemble methods:

- In **averaging methods** or parallel methods, the principle is to build several estimators independently and then to average their predictions. On average, the combined estimator is usually better than any of the single base estimator because its variance is reduced.

- Conversely, in **boosting methods** or sequential methods, base estimators are built sequentially and one tries to reduce the bias of the combined estimator. The motivation is to combine several weak models to produce a powerful ensemble.

Within ensemble methods there are many techniques. For this work, we have selected Random Forest, Extra Tree and Adaboost algorithms completing both averaging and boosting methods.

2.2.3.1.. Random Forest algorithm

The Random Forest algorithm belongs to averaging methods. As the name suggest, this algorithm creates the forest with a number of decision trees. In general, the more trees in the forest the more robust the forest looks like and higher accuracy in the results is obtained.

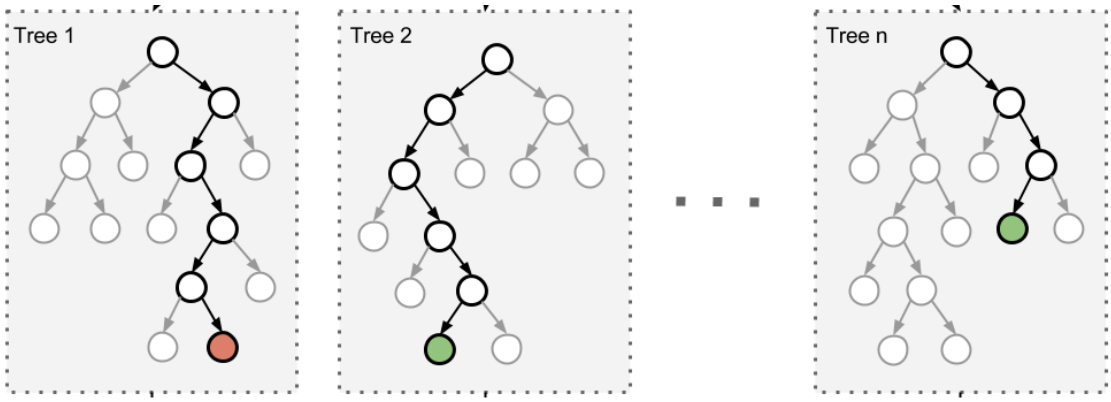

Figure 2.4: The Random Forest operation.

Similarities between the Decision Tree algorithm and Random Forest are obvious, even though we will see that with Random Forest the accuracy improve markedly. In order to explain the process, let us suppose that our initial data have m features, which is the total number of characteristics like magnitudes, coordinates, proper motions and so one, in our simulated sample case. Two parts in the operation code of the algorithm can be distinguished. Through them, we are going to explain how Random Forest works.

1. **Random forest pseudocode**.

   a) Randomly selects s features from a total of m features, where $s << m$.

   b) Among these s features, compute the node d using the best split point. Then, this node is split again into daughter nodes using the best split. It works following the rules of decision trees.

   c) Repeat these two steps until k number of nodes has been reached.

   d) Build a forest of n number of trees by repeating steps 1 to 3 n times as Fig. 2.4 shows.

2. **Pseudocode to perform predictions from the created random forest classifier.**

   (e) Take the test features and use the rules of each randomly created decision tree to predict the outcome and stores the predicted outcome (target).

   (f) Calculate the votes for each predicted target.

(g) Consider the high voted predicted target as the final prediction from the random forest algorithm.

One of the biggest advantages of random forest over decision tree is the algorithm on which the former one works i.e. Bagging algorithm. It means random forest replaces the data/population used to construct the tree and also the explanatory variables are bootstrapped so that partition is not done on the same important variable. This fact is very important since decision trees do not follow this algorithm. It simply keeps on building trees by determining the important variable which depends on homogeneity. Bootstrapping reduces bias and variance making the model more robust and accurate.

2.2.3.2.. Extra Tree algorithm

The main objective of the Extra-Tree method is going further randomizing tree building in the context of numerical input features, where the choice of the optimal cut-point is responsible for a large proportion of the variance of the induced tree.

With respect to random forests, the method drops the idea of using bootstrap copies of the learning sample, and instead of trying to find an optimal cut-point for each one of the s randomly chosen features at each node, it selects a cut-point at random. This idea is rather productive in the context of many problems characterized by a large number of numerical features varying more or less continuously: it leads often to increased accuracy thanks to its smoothing and at the same time significantly reduces computational burdens linked to the determination of optimal cut-points in standard trees and in random forests.

From a statistical point of view, dropping the bootstrapping idea leads to an advantage in terms of bias, whereas the cut-point randomization has often an excellent variance reduction effect.

2.2.3.3.. Adaboost algorithm

The Adaboost algorithm belongs to boosting methods. Boosting has been a very successful technique for solving the two-class classification problem. In going from two-class to multi-class classification, most algorithms have been restricted to reducing the multi-class classification problem to multiple two-class problems. Since in our problem we have three classes, we are going to explain those one which combine the elements for a better classification in this sense.

The algorithm focuses on classification problems and aims to convert a set of weak classifiers into a strong one. The final equation for classification can be represented as

$$F(x) = \sum_{m=1}^{M} \theta_m f_m(x) \tag{2.3}$$

where f_m stands for the m_{th} weak classifier and θ_m is the corresponding weight. It is exactly the weighted combination of M weak classifiers. The whole procedure of the AdaBoost algorithm can be summarized as follow.

1. Initially, an identical weight (w_i) is assigned to all the data of the training set, where n is the size of the dataset.

$$w(x_i, y_i) = \frac{1}{n}, i = 1, 2, 3, ..., n. \tag{2.4}$$

2. The model is fitted using the training set.

3. The error of this model is computed counting how many points has been misclassified and then, they are identified. In that cases, the weight is increased as

$$\theta_m = \frac{1}{2} ln(\frac{1 - \varepsilon_m}{1 + \varepsilon_m}) \tag{2.5}$$

4. From that point, a new model is fitted using the set of modified weights.

5. Steps 3 - 4 are repeated.

6. Final model: voting weighted is done by the weights of all the models. By this way, equation 2.3 is trained and updated using the different weights computed as the previous steps.

In Figure 2.5 we show a practical example. For a very small dataset, $+$ (plus) and $-$(minus), the Adaboost algorithm assigns equal weights to each data point as specified in step 1. Then it applies a decision stump to classify them. The decision stump (D1) has generated a vertical line at the left side to classify the data points. Although, this vertical line has incorrectly predicted three + (plus) as – (minus).

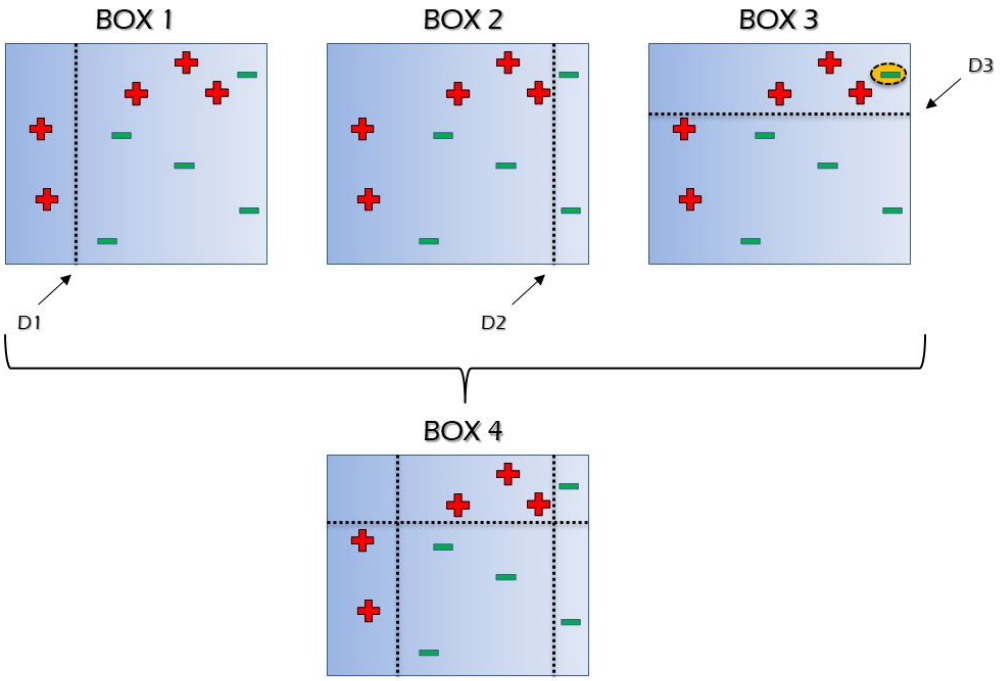

Figure 2.5: Example of how Adaboost algorithm works showing the evolution of a small dataset according to their weights.

In such case,the algorithm will assign higher weights to these three + (plus) and apply another decision stump. Here, you can see that the size of three incorrectly predicted +

(plus) is bigger as compared to rest of the data points. Therefore, the second decision stump (D2) will try to predict them correctly. Now, a vertical line (D2) at the right side of this box has classified three miss-classified + (plus) correctly. But again, it has caused miss-classification errors. This time with three -(minus). Again, the algorithm will assign higher weight to three – (minus) and apply another decision stump. Now, three – (minus) are given higher weights. A decision stump (D3) is applied to predict these miss-classified observation correctly. This time a horizontal line is generated to classify + (plus) and – (minus) based on higher weight of miss-classified observation. From here, a combination of D1, D2 and D3 have been formed to obtain a stronger prediction. The algorithm has classified these observation quite well as compared to any of individual weak learner.

2.3.. Choosing the best algorithm

In this section we will select the best algorithm candidate for evaluating and classifying our simulated data among the five machine learning algorithms previously exposed. Naturally there are other methods that they have not been mentioned. However, the algorithms considered here are representative of the supervised branch, reason why we expect that the results applied to our simulated data are going to be accurate.

2.3.1.. Previous considerations

For choosing the best algorithm for our classification data purpose, we will compare different classification aspects that will provide us with the necessary information in order to decide which is the best solution. Some of the aspects that in any classification algorithm should be considered are defined as follow:

- **Accuracy**: the number of correct predictions from all predictions made. This is the central parameter in any classification method.
- **Training time**: the number of seconds, minutes or even hours necessary to train a model. It strongly depend of the algorithm. Training time is often closely tied to accuracy, one typically accompanies the other. In addition, some algorithms are more sensitive to the number of data points than others. When time is limited it can drive the choice of algorithm, especially when the data set is large.
- **Over-fitting**: refers when a model fits the training data too well. This happens when a model learns the detail and noise in the training data to the extent that it negatively impacts the performance of the model on new data. This means that the noise or random fluctuations in the training data is picked up and learned as concepts by the model. The problem is that these concepts do not apply to new data and negatively impact the models ability to be generalized.
- **Parameters**: Parameters are the knobs that a data scientist gets to turn when setting up an algorithm. These parameters are generally numbers that affect the algorithm's behavior, such as error tolerance or number of iterations, or options between variants of how the algorithm behaves. The training time and accuracy of the algorithm can sometimes be quite sensitive to getting just the right settings. Typically, algorithms

with a large number of parameters require a precise trial and error test to find the best fit. Algorithms with many parameters typically indicates that the algorithm has greater flexibility and thus it can often achieve very good accuracy. Examples of parameters that we have seen in this work are the umber of decision trees in a Random Forest algorithm, the number of splitting levels in the same one, the number of neighbours (K) in the KNN algorithm, etc.

- **Features**: for certain types of data, the number of features can be very large compared to the number of data points. This is often the case with genetics or textual data. The large number of features can hinder some learning algorithms, making training time unfeasibly long.

These five basic characteristics have been used to indicate the best method to keep our trained model for future predictions. Thus, through these aspects we are going to elaborate some statistics for every machine learning algorithm considered in this work so as to contrast results and choose which is more adequate for our purpose.

2.3.2.. Training the algorithm: splitting the data

The first step, whatever supervised method explained before used (Random Forest, KNN, Adaboost, Decision Tree or Extra-Tree), it is to train the algorithm. This is done by splitting the data in two subsamples, one properly for training and the other one for testings. The complete flux diagram of how supervised algorithm works is shown in Figure 2.6.

Just for remembering, in Chapter 1 we extracted a number of white dwarf candidates through the enormous sample from the IGSL catalogue which we designated as observed sample. For this sample we ignore to which population (thin, thick or halo) these stars belongs, thus being our purpose to classify them. Afterwards, we simulated a sample of 165000 even though we are going to consider only those synthetic white dwarfs under the cooling sequence limit of $V_{TAN} = 20km/s$ as Fig. 1.6 shows (we have now 136000 synthetic white dwarfs instead of the total amount of 165000). The stars of this simulated sample are labelled, so we perfectly know to which population they belong. Now, the supervised algorithm starts. Firstly, it split the simulated sample in two samples, as previously stated: the training and the testing sample. Typically the training sample contains 95000 (70%) objects and the testing sample around 41000 (30%) objects. Secondly, the training sample of 95000 white dwarf is configured using two matrices: one just for features (magnitudes, coordinates, proper motions,...); and the another one for labels of each star, in our case it would be halo, thick or thin.

Once we have the algorithm trained, we will use the other sub-sample for testing our classification model. (see figure 2.6). This classification model is applied to the X test matrix in order to verify and analyse the efficacy of our classification model already trained, which is the same matrix of the training sample with the same features but with different white dwarf objects. When we apply the algorithm, we are able to predict the class of the remaining 41000 white dwarfs. So, in this case, the Y test matrix will serve us to verify if this stars are well-classified or not. Finally, we will compute the accuracy of the classification model and compare it among different algorithms.

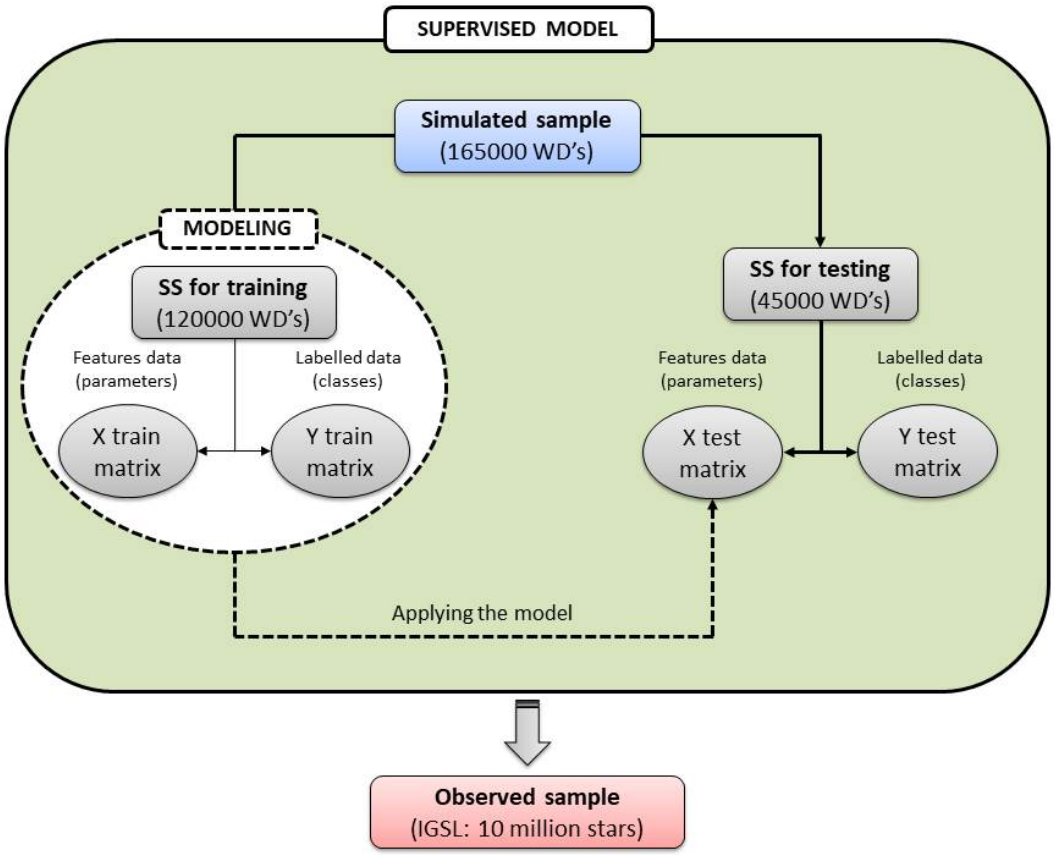

Figure 2.6: Operational configuration of an supervised model.

2.3.3.. Accuracy

In this section the accuracy has been derived for each algorithm studied in this work. It is important to remark that these results are specifics for our simulated sample given that the performance of a certain classification algorithm depends of the characteristics of the data under study.

In Figure 2.7 we represent the accuracy in front of the number of estimators for Random Forest, Extra-Tree, Adaboost and KNN algorithms. By estimators we mean the number of decision trees both Random Forest and Extra-Tree can have; the number of recalculated weights for Adaboost; or the total number of K-neighbors for KNN. Two types of lines are plotted in Fig. 2.7: the dashed-line which refers to results of training samples, that is compare the X train matrix with the Y train matrix (labels) after the classification model is fitted at least one time; and the continuous line where compares the classification of the X test matrix using the model trained before with the real classification of the same X test matrix represented in the Y test matrix. (See again the figure 2.6).

At first glance of Fig. 2.7 shows that the general tendency is that the accuracy increases as more decision trees are employed up to a certain value where the accuracy is maintained constant for all four algorithms. Note that this plot has not sense in the case of the Decision Tree algorithm since it only works with a single decision tree (estimator).

If we focus on the test results (continuous lines), which they give the important information since they represent the real prediction of the model, we become aware that both Random

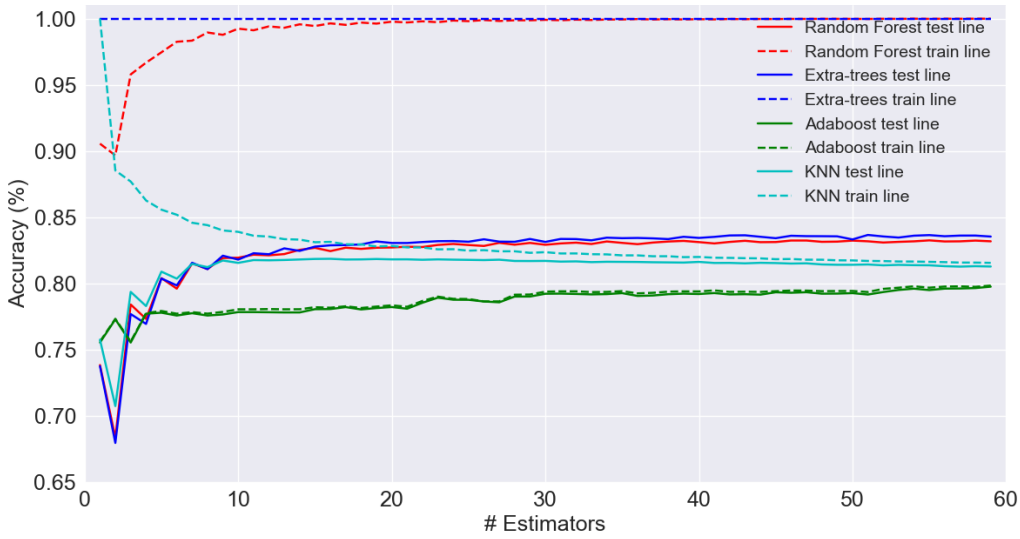

Figure 2.7: Accuracy achieved by Random Forest, Extra-Tree, Adaboost and KNN algorithms applied to the simulated sample in terms of the corresponding number of estimators.

Forest and Extra-Tree reach the best accuracy, with maximum values of 83% and 83.3%, respectively. These values are achieved from 30th decision tree. On the other hand, Adaboost is more constant regardless the number of estimators, reaching a maximum accuracy of 80%. Besides, KNN algorithm achieves a maximum precision of 81%, which is quite acceptable.

Regarding training results (dashed lines), they show a completely different behaviour We see from Fig. 2.7 that only Adaboost algorithm follows exactly the same behavior and maintains the same accuracy as in the test results. However, both Random Forest and Extra-Tree algorithms increase the accuracy notably in comparison with their test results. As remarkably aspect, a 100% accuracy has been reached by the Extra-tree algorithm. This signify that our model is able to predict without any mistake. But if we observe what happen when the model is translated to the test data, we obtain a 83% instead of that 100%. Something similar happens with the Random Forest even though its training line starts from an accuracy of 91%. This phenomena is called over-fitting and it does not appear in the cases of KNN or Adaboost algorithms.

As previously stated, over-fitting is the tendency of most Machine Learning algorithms (specially those ones which use decision trees) to adjust them to very specific characteristics of training data that they have no causal relationship with the objective function we are looking for to generalize. In other words, a model is going to be over-fitted when we see that it performs well with the training data, but its accuracy is notably lower with the evaluation data. This is because the model has *memorized* the training data and it is not able to generalize the rules to predict the new data. Thus, if we are able to avoid this phenomena in our training, for sure we were going to reach a better accuracy in our test data. The question is how can we do it? There are several ways to avoid the over-fitting such as dividing the training data in some sets (cross-matching method), or also adjust the initial parameters of the algorithm in order to decrease the accuracy of the training sample. Finally, it is also possible to increase the amount of the initial data. This would

cause a restarted effect modeling the data and probably this training score of 100% would be reduced. In this work, we have focused for changing parameters until the maximum possible test accuracy avoiding the over-fitting.

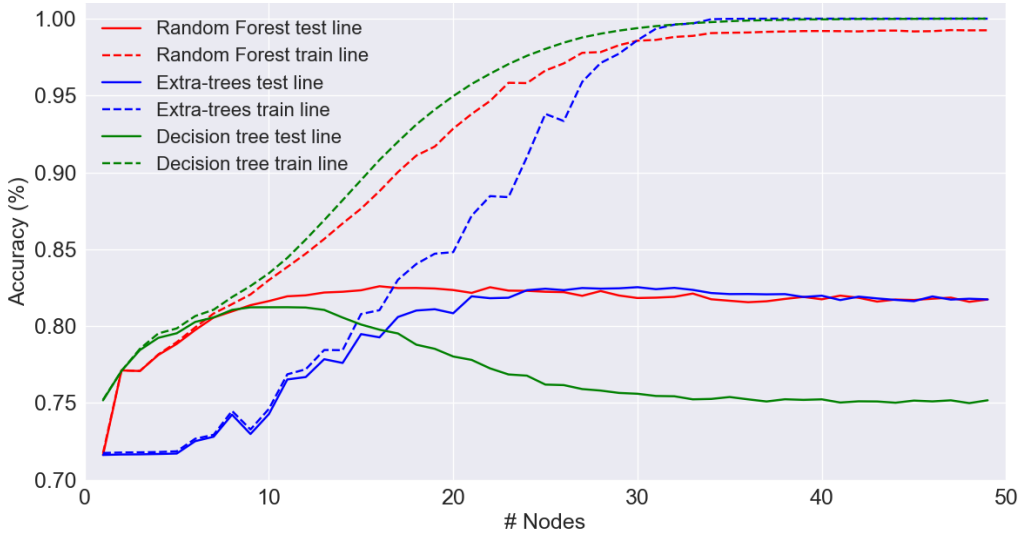

Figure 2.8: Accuracy achieved by Random Forest, Extra-Tree and Decision Tree algorithms for simulated sample in terms of the number of nodes when the algorithm is fitted.

In Figure 2.8 we present the accuracy for each model as a function of the number of levels that every decision tree have, or in other words, their maximum depth. This is one of the main factors that provokes a certain over-fitting in decision tree models. Training and testing lines are plotted in Fig. 2.8 as continuous and dashed lines, respectively. First of all, we see that the maximum precisions are achieved in the testing line for an 8 nodes Decision Tree, 16 nodes for Random Forest and 30 nodes for Extra-Tree. For higher number of nodes, models loss in generalization and they begin to be over-fitted since training lines ascend quicker towards an accuracy of approximately 100%. This fact permit us to derive the maximum depth for each model in order to avoid the over-fitting.

2.3.4.. Training time

Training time is the other important parameter that characterize the Machine Learning process. In this work we have determine which of the five algorithms under study spend more time training our simulated sample. By means of Figure 2.9 the time rate for each star sample is plotted.

The first important thing observed in Fig.2.9 is that the training time increases linearly for all the algorithms under consideration, or at least, that is what it seems in the range under study. Consequently, we can stated that all the algorithm are, in principle, scalable to larger samples. Secondly, we realize that the slowest algorithm is Random Forest algorithm. In contrast, KNN algorithm is the quickest one. For a sample of 95000 white dwarfs, Random Forest training time spends $18.5\,$s while KNN takes only $0.4\,$s. Despite the relative difference is quite large between these two models, the absolute value of the training time in terms of computational time is acceptable for all the algorithms.

Figure 2.9: Training time for different number of stars according to each algorithm.

2.3.5.. Final comparison and conclusions

For the five algorithms under study we have recalculated them with the optimal estimators in order to obtain the highest accuracy as well as to try to avoid over-fitting effects. The corresponding results showing the percentages of each classified population, accuracies and training time for each algorithm is shown in Table 2.1.

	Thick (%)	Thin (%)	Halo (%)	Training accuracy (%)	Test accuracy (%)	Training Time (s)
DT	0.14	0.78	0.08	0.82	0.81	1.8
RF	0.11	0.83	0.06	0.89	0.85	18.5
ET	0.08	0.85	0.07	0.95	0.85	6.2
KNN	0.09	0.83	0.08	0.82	0.82	0.4
AB	0.08	0.84	0.08	0.80	0.80	16.2

Table 2.1: Results showing the percentages of each classified population, accuracies and training time for each algorithm (DT: Decision Tree, RF: Random Forest, ET: Extra-Tree, KNN: k-Near Neighbours, AB: Adaboost).

First of all, from the results of Table 2.1. we observe higher test accuracies while training accuracies are reduced. This implies that, in general, there is not over-fitting. A training accuracy between $80-90\%$ is quite correct. Although in the case of Extra-tree is not like that.

Secondly, the Table 2.1. shows the population distribution made by each method. We verify that most of the white dwarfs on the simulated sample are in the thin disk followed by the thick disk and finally, the halo region. This result, as well as their percentages are in agreement with the observational results of the different proportion of the components of the Galaxy. So, we can conclude that all the algorithms studied perform a proper classification of the data.

In order to choose the best algorithm for our purpose we need to take into account the one that achieve the highest possible accuracy for a more feasible classification. Training time is also important even though it is is secondary for our goals. Since we will be treating with 136000 white dwarfs (total size of simulated sample), our results (see Fig.2.9) indicate that the training time for 95000 (training simulated sample) is in the worst case around a few seconds, which we consider it completely acceptable.

Consequently, we reject automatically those methods with the lowest accuracy. Thus, our best classification algorithms result to be the Random Forest algorithm and the Extra-Tree algorithm. Actually, as previously explained, they work in a very similar way. After modifying parameters as well as when trying to avoid over-fitting, we become aware that the cross-matched method is needed to increase both Random Forest and Extra-Tree algorithms. Although a 85% of well-predicted cases is more than satisfactory. The difference between this two models is quite small. However Extra-Tree has a training accuracy of 95% what means that the algorithm wrongly classify only a 5% of cases, while Random Forest has a 89% of training accuracy which is more reasonable. This values indicate us that Extra-Tree is still affected by some residual over-fitting, and previously said, we would need the cross-matching for Extra-Tree. For all the reasons above exposed, we choose **Random Forest** as our best classification algorithm.

CHAPTER 3. APPLYING THE RANDOM FOREST TO THE SIMULATED SAMPLE

In this chapter we present the results of applying the Random Forest algorithm to our simulated sample. The process so far have been to prepare an optimal machine learning algorithm in order to be trained in a simulated sample of white dwarfs. Eventually this classifying algorithm will be applied to other sample, in particular the observed sample provided by Gaia future releases. As seen in previous chapters, Random Forest algorithm is presented as the best choice among these strategies, obtaining a well maximum accuracy of $85\,\%$.

In this chapter, some statistical results of applying the Random Forest to the simulated sample are showed. Through these results, we are able to visualize the behaviour of the classification model providing us interesting aspects about the learning process. Besides on this, we are going to check how the Random Forest classify our particular labeled data and compare its results with the classical reduced proper motion diagram method employed so far in star classification.

3.1.. Random Forest in the simulated sample

3.1.1.. The structure of the classification model

Our classification model follows a Decision Tree structure splitting the data by means of different valuations. The Random Forest algorithm use some decision trees to build a complex way to classify the data instead of the Decision tree algorithm, which only uses one tree. Here is the reason why a better accuracy is achieved using one instead of another.

The working structure of one simple decision tree is showed in Figure3.1. All black spots in the top panel of Fig.3.1 compose one single decision tree of all possible decision trees that conform our Random Forest. For clarity reasons only one decision tree is shown intead of the complete algorithm structure in Fig.3.1. In order to better visualize the content of the tree, consecutive zooms are executed in just one section of the structure. Basic elements of the decision tree (see again section 2.2.1) are viewed, like different nodes and also arrows connecting each node. Every node shows one or more parameters which are split in that level as well as the number of samples contained in this node, and finally, which is the class in that depth.

As an example, the Fig.3.1 shows one purple node in which all stars will be split with the condition of mass $\leq 0.789\,M_{\odot}$. It also states that unless 112 stars reach this level, being stars from the thin disk (class). Note that this node is in 4^{th} level depth and there are only 112 stars when initially we had 95000 stars. This means that in the firsts levels stars have probably been distributed very non-uniformly.

In order to comprehend how Random Forest works we need to imagine several independent sequences like the one presented in Fig.3.1. Every time we train the data, the distribution of these sequences is completely different. In other words, if one particular star

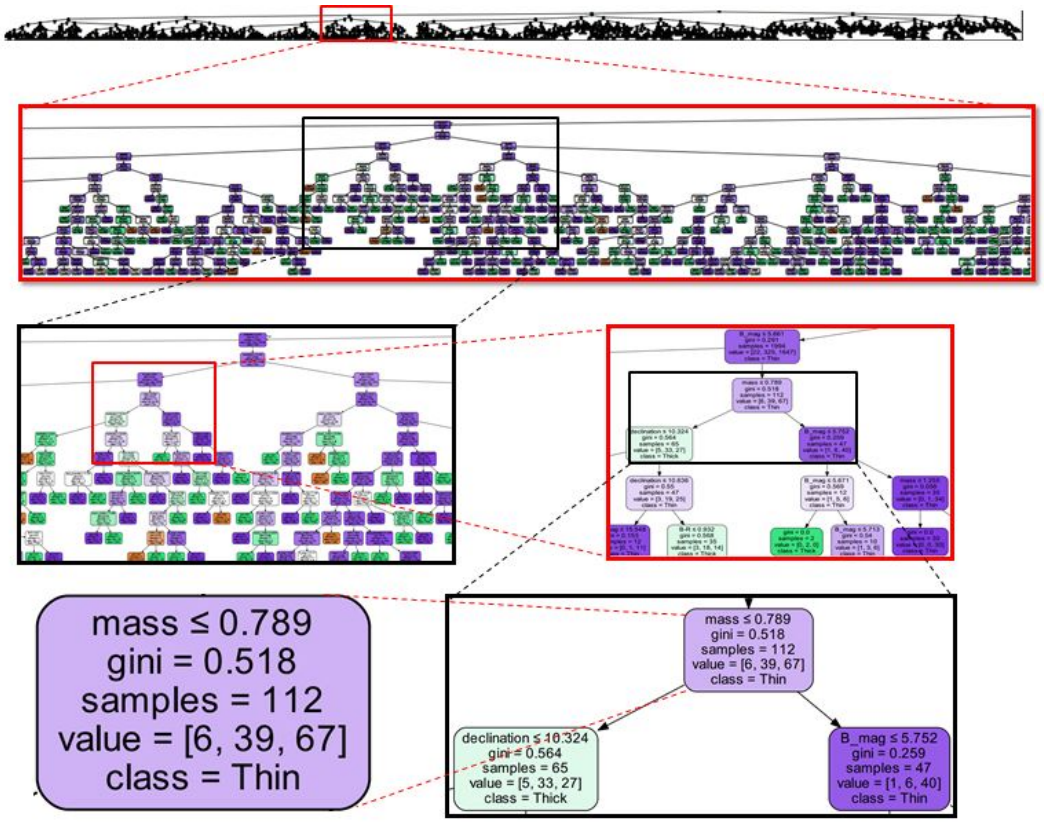

Figure 3.1: One typical decision tree out of all possible decision trees that conform the Random forest algorithm. Colors on boxes make reference to the three categories that stars are classified: thin, thick and halo.

tends to choose one simple path among all this trees, this same star will probably goes for a different path using other trained Random Forest model. The final classification of this star will be the same, obtaining consequently the same accuracy of the classification model, but the particular distribution have not to be always the same.

3.1.2.. Importance of the features

A benefit of using boosting methods is that after the boosted trees are constructed, it is relatively straightforward to retrieve importance information for each attribute or feature of the sample. Generally, *importance* provides a score that indicates how useful or valuable each feature was in the construction of the boosted decision trees within the model. The more an attribute is used to make key decisions within decision trees, the higher its relative importance. This importance is calculated explicitly for each attribute in the sample, allowing attributes to be ranked and compared to each other.

Importance is calculated for a single decision tree by the amount that each attribute split point improves the performance measure, weighted by the number of observations the node is responsible for. The performance measure may be the purity (Gini index) used to select the split points or another more specific error function. This Gini index is indicated in each node in Fig.3.1. The feature importances are then averaged across all the decision trees within the Random Forest.

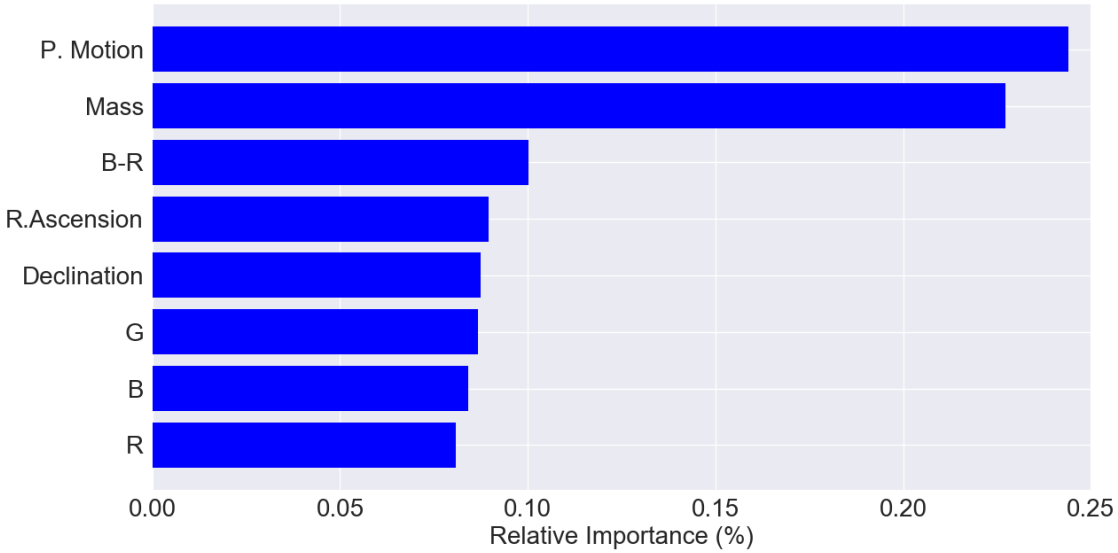

Figure 3.2: Relative importances for each attribute of the simulated sample.

These features importances are plotted in Figure3.2. The first thing we observe is that both proper motion and mass are the most important features that the algorithm uses to classify the data. In contrast, absolute magnitudes are the less used of them.

3.1.3.. Confusion matrix

Once we have seen how the algorithm works and which are the most important features or attributes used for building the Random Forest, we will see now the basic statistics about the number of white dwarfs classified after the prediction. The confusion matrix is introduced as a really powerful tool to understand better the classification methodology in supervised models. The terminology employed by the confusion matrix contains the next four possible outcomes:

- True positives (TP): These are cases in which the algorithm predict the star like a particular class, for instance, a halo star, and it naturally belongs to halo class.
- True negatives (TN): The algorithm predict that this star does not belongs to halo class, and in effect it does not belong to this class.
- False positives (FP): The algorithm predict that this star belongs to a particular class such as halo, but the star does not belongs to this class.
- False negatives (FN): There are cases in which the algorithm predict that the star does not belongs a particular class even though it belongs to this class.

For an ideal classification algorithm, the confusion matrix should appears as an identity matrix. As the classification algorithm departs from the ideal case, elements out the diagonal start to be different from zero. In other words, it is important that the algorithm

recognizes the stars in its true region (TP) as well as that it recognizes those stars that do not belong to a specific region (TN). Both are equally important for this kind of algorithms. Through these parameters, we are able to calculate the accuracy of each label as:

$$Accuracy = \frac{TP+TN}{TP+FP+TN+FN} \tag{3.1}$$

Although accuracy is not a reliable metric for the real performance of a classifier, because it will yield misleading results if the data set is unbalanced (that is, when the numbers of observations in different classes vary greatly). For instance, if there were a 95% of thin white dwarfs and only a 5% of halo white dwarfs in the sample, a particular classifier might classify all the observations as thin white dwarfs. The overall accuracy would be $95\,\%$, but a more detail analysis will show that the classifier would have a $100\,\%$ recognition rate for thin stars but a null $0\,\%$ recognition rate for the halo class. For that reason, it is very recommendable to build a great, uniform and well-distributed sample to train the model. This fact explains the complex process followed in chapter 1 to build a correct sample (the simulated sample) to train and test our algorithm.

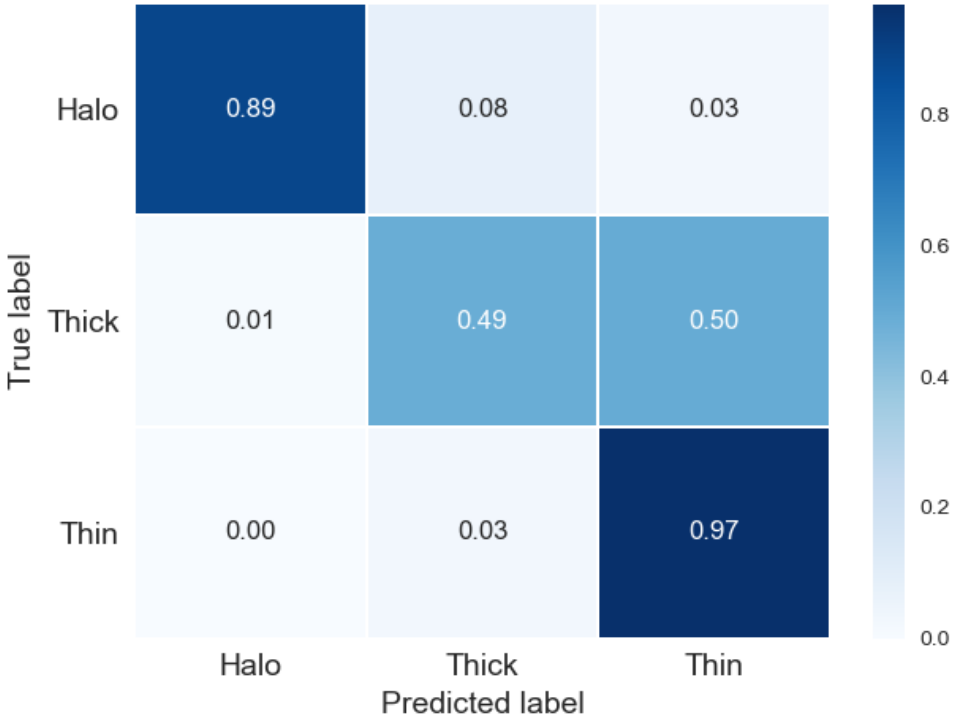

Figure 3.3: Confusion matrix of the Random Forest algorithm applied to the test simulated sample.

The Figure 3.3 shows the confusion matrix of our Random Forest applied to the X test matrix (34610 objects) of the test sample (see again Fig.2.6) extracted from our simulated sample. The matrix axis, ordinate and abscissas, represent the true class and the predicted class, respectively. Besides, each axis have three possible labels corresponding to the three populations: thin, thick and halo. Each cell of the confusion matrix represents the

number of white dwarfs of a certain population classified in a certain class. Additionally, a color bar is inserted in the right side of Fig.3.3 proportional to the number density of stars.

The most populated group belongs to the thin disk population. Four outcomes of this class are represented in Table 3.1. We clearly see that the amount of TP overcomes greatly the other ones. This indicates that the algorithm predicts correctly 23271 white dwarfs in the thin region over the total white dwarfs that really belongs to this population, that it is a number of 24093 white dwarfs (TP + FN). Of this total amount, the algorithm predicts erroneously that 808 white dwarfs are in the thick region and 14 are in the halo. These 822 stars are FN. At the same time, we see that there are 10517 white dwarfs (FP + TN) that they do not belongs to the thin class and however, the algorithm identify 3724 white dwarfs of this total as stars pertaining at this class. Also it classified 6793 white dwarfs that do not belong to the group. Thus, we see that generally speaking, the algorithm produces a good prediction of the thin class and taking into account equation 3.1, it obtains an specific accuracy of 86%.

		True label	
		Thin	Non-thin
Predicted label	Thin	23271 TP	3724 FP
	Non-thin	822 FN	6793 TN

Table 3.1: Four outcomes for the thin population extracted from the confusion matrix.

Results for the thick disk are presented in Table 3.2. The algorithm correctly classified 3586 white dwarfs (TP) of the total real number of 7358 stars (TP + FN) of this population, misclassifying 3772 white dwarfs like non-thick. Actually we see from the confusion matrix that 3615 stars of the thick disk are misclassified as thin and 157 are misclassified as halo. Thus, a big portion of thick white dwarfs are erroneously classified as thin stars. There are 27252 white dwarfs in non-thick class (FP + TN) and 1051 of which are identified as thick. Thus we see that the number of TP is much smaller than in the thin case and therefore, the prediction of correct white dwarfs is really poorer than the previous case but the prediction of false white dwarfs is pretty well. Thus, the thick population obtains an specific accuracy of 85%.

		True label	
		Thick	Non-thick
Predicted label	Thick	3586 TP	1051 FP
	Non-thick	3772 FN	26201 TN

Table 3.2: Four outcomes of the thick population extracted from the confusion matrix.

Finally, we can see the four outcomes of the halo population showed in Table 3.3. Primarily, we observe a really small value of 2807 TP which is smaller than the previous one. The total real value of halo stars is 3159 (TP + FN) and the algorithm only misclassified 352 of these like non-halo white dwarfs. According to the confusion matrix as well, 243 are in the thick region and 109 are in the thin region of this total 352 white dwarfs. There are 27252 white dwarfs non-halo class (FP + TN) and 109 of which are identified with the halo label. Note that in this case, there are 31280 white dwarfs (TN) that are good non-classified. So, the specific accuracy of this label is 98%.

		True label	
		Halo	Non-halo
Predicted label	Halo	2807 TP	171 FP
	Non-halo	320 FN	31280 TN

Table 3.3: Four outcomes of the halo population extracted from the confusion matrix.

In view of the statistical results obtained so far, we can conclude that there is a conflict between white dwarfs belonging to the disk population, both the thin and the thick regions since they are not completely distinguished. There are few halo stars but the error committed in its classification is smaller than in the case of the disk stars. Although both population have achieved a well general accuracy of $85\,\%$, it is clear that exist a intrinsic difficulty in disentangle both disk populations. This is corroborate by the confusion matrix of Fig.3.3 where we observe that the values outside the diagonal are larger for the thin and thick populations. On the other hand, halo population, even the initial smaller proportionality of this objects, is perfectly identified by our classification algorithm.

3.1.4.. Predicted RPM diagram

Once the classification model have been detailed, tested and studied in depth, we can derive the RPM diagram for our simulated sample according to the predicted classification of the algorithm. However, previous to this, we want to show the real RPM diagram four our simulated sample. This is in done in Figure3.4 where we show the RPM diagram for the simulated sample. In this diagram stars are plotted according to the population provided by our simulator, this is what we call the true or real classification of the white dwarf sample. Also, for comparative purposes we show in Fig.3.4 the extrapolated lines for tangential velocities $V_{\mathrm{TAN}} = 20,\ 100$ and $200\,\mathrm{km}\cdot\mathrm{s}^{-1}$. These lines correspond, as we will later see, to the selection region associated to the thin, thick and halo population, respectively.

Now, we can apply our Random Forest algorithm to the simulated sample. In an ideal case we will retrieve the diagram showed in Fig.3.4. The results obtained by our classification algorithm are plotted in Figure 3.5. The resemblance with the real classification showed in Fig.3.4 is more than acceptable, provided that the accuracy of our algorithm has been estimated in a 85%. There exist a small portion of withe dwarfs misclassified, mainly thin and thick disk stars. This fact can be visualized in the Fig.3.5 as less blue dots (thick stars) when compared with the same region of Fig.3.4 where coexist with red dots (thin stars). This is reasonably because this two population have closer characteristics in front of halo population. On the other hand, halo stars are clearly identified since they are more isolated at larger reduced proper motion values.

3.2.. RPM diagram selection method

The standard procedure to classify white dwarfs into its different populations has been since many years ago the use of the RPM diagram. As explained along this work the delimiting lines as a function of the tangential velocity in the RPM diagram permit us to classify a white dwarf according to the region where the star is located. This method has

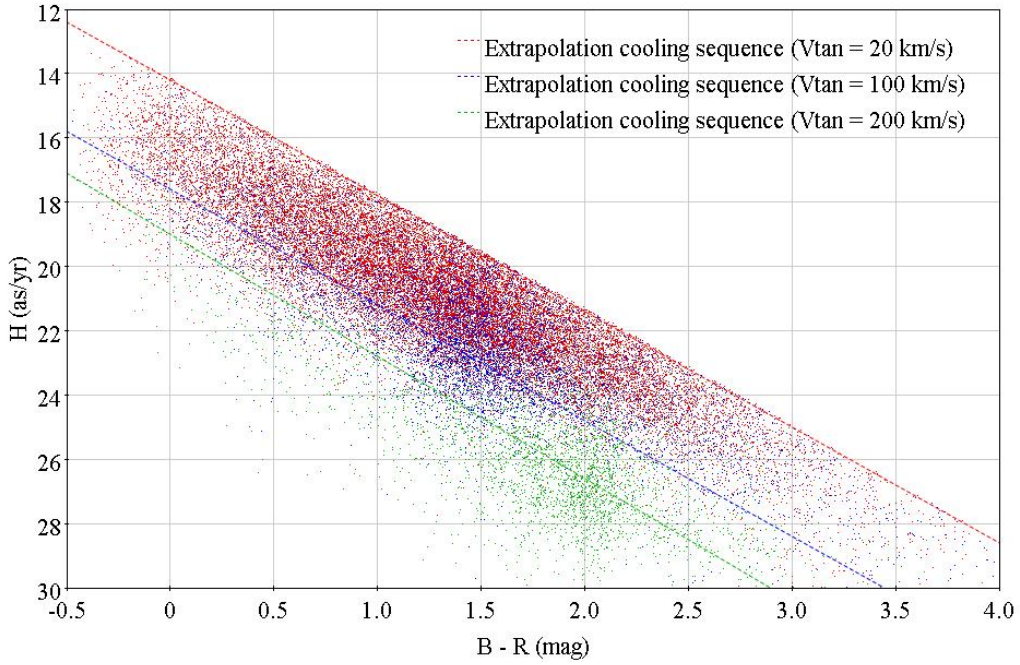

Figure 3.4: RPM diagram of the simulated sample showing the real distribution into the three different populations.

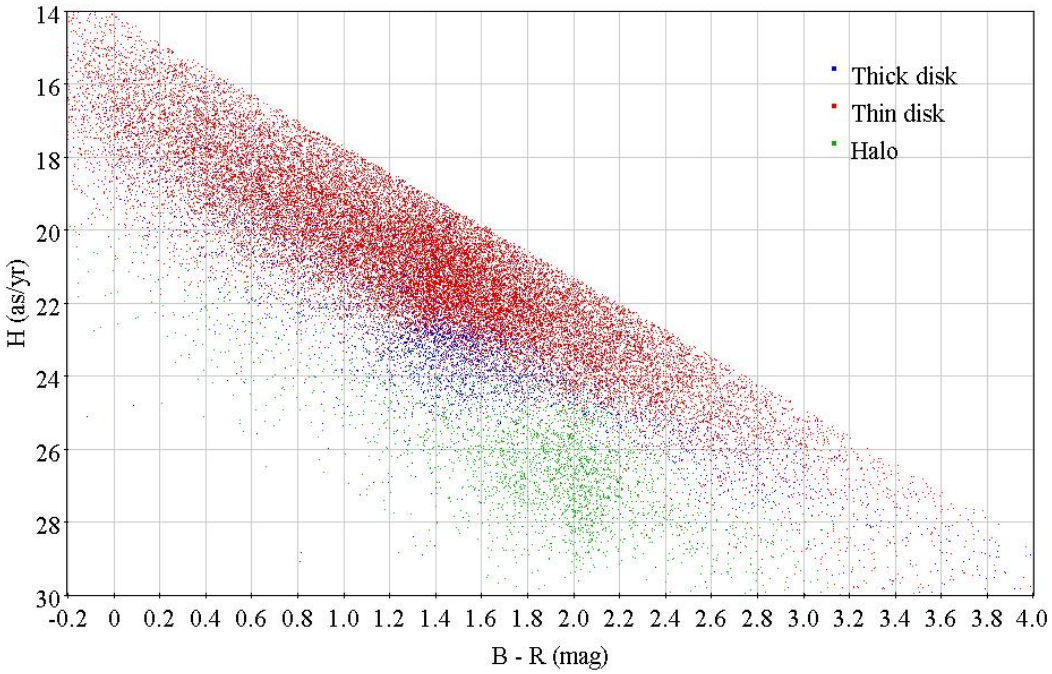

Figure 3.5: RPM diagram showing the predicted classification by our Random Forest algorithm.

been employed recently in the classification of the SuperCosmos survey [8]. Typically, stars located between the delimiting lines for $V_{\mathrm{TAN}} = 20\,\mathrm{km}\cdot\mathrm{s}^{-1}$ and $V_{\mathrm{TAN}} = 100\,\mathrm{km}\cdot\mathrm{s}^{-1}$ are considered thin disk stars, those located between $V_{\mathrm{TAN}} = 100\,\mathrm{km}\cdot\mathrm{s}^{-1}$ and $V_{\mathrm{TAN}} = 200\,\mathrm{km}\cdot\mathrm{s}^{-1}$, as thick stars, and finally, beyond $V_{\mathrm{TAN}} = 20\,\mathrm{km}\cdot\mathrm{s}^{-1}$ are classified as halo white dwarfs. This method, obviously, disregard other features of the sample, given that

only take into account the location of the star within the reduced proper motion versus colour diagram. Conversely, machine learning algorithm, and in particular Random Forest algorithm, take into account the whole space of parameters and not just a projection onto a two dimensional fold.

The classification obtained by the RPM diagram selection method is shown in Figure 3.6. As we can check, the distribution of white dwarfs in this diagram is given only by the location between selection lines of different tangential velocities. Clearly, this procedure ignore the possibility that two or more populations should be superimposed in the RPM diagram, such is the case of the thin and thick disk stars.

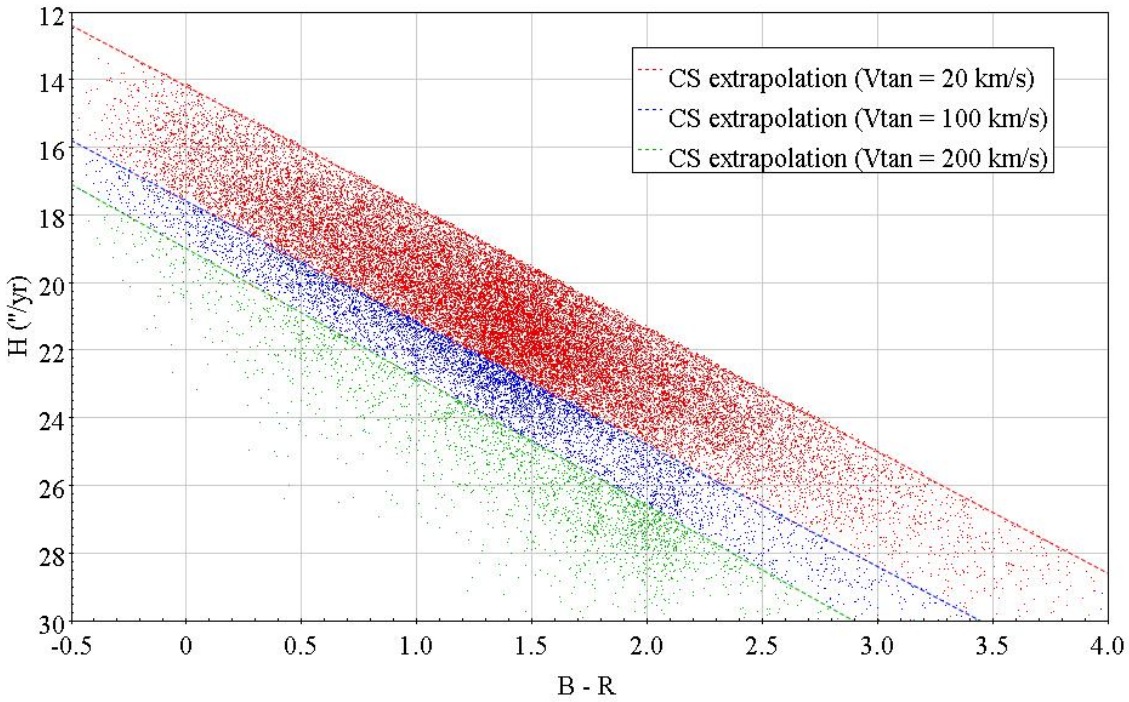

Figure 3.6: RPM diagram classification of white dwarfs when applied the classical method of selecting objects.

Just comparing Fig.3.6 with the true distribution displayed in Fig.3.4 it is clear that the error committed with the classical RPMD diagram method is really huge. There exist a lot of thick white dwarfs that are confused as thin ones, even halo stars also. The same occurs for the thick region where there is an important number of halo stars that are confused as thick ones, and also some thin white dwarfs. Consequently the mutual contamination among population is quite evident when applied this method.

3.2.1.. Confusion matrix for the RPM diagram selection method

By means of the confusion matrix we are going to quantify the qualitative result obtained in the previous section. We will follow the same procedure in deriving the confusion matrix as did with the Random Forest algorithm. Finally, both confusion matrices will be compared.

The new confusion matrix, obtained after applying the RPM diagram selection method is showed in Figure 3.7. First, we check that this new confusion matrix departs from a

diagonal identity matrix, that would be the ideal case. The confusion among the different populations has increased with respect Fig.3.3, as corroborate for the large values of Fig.3.7 outside the diagonal.

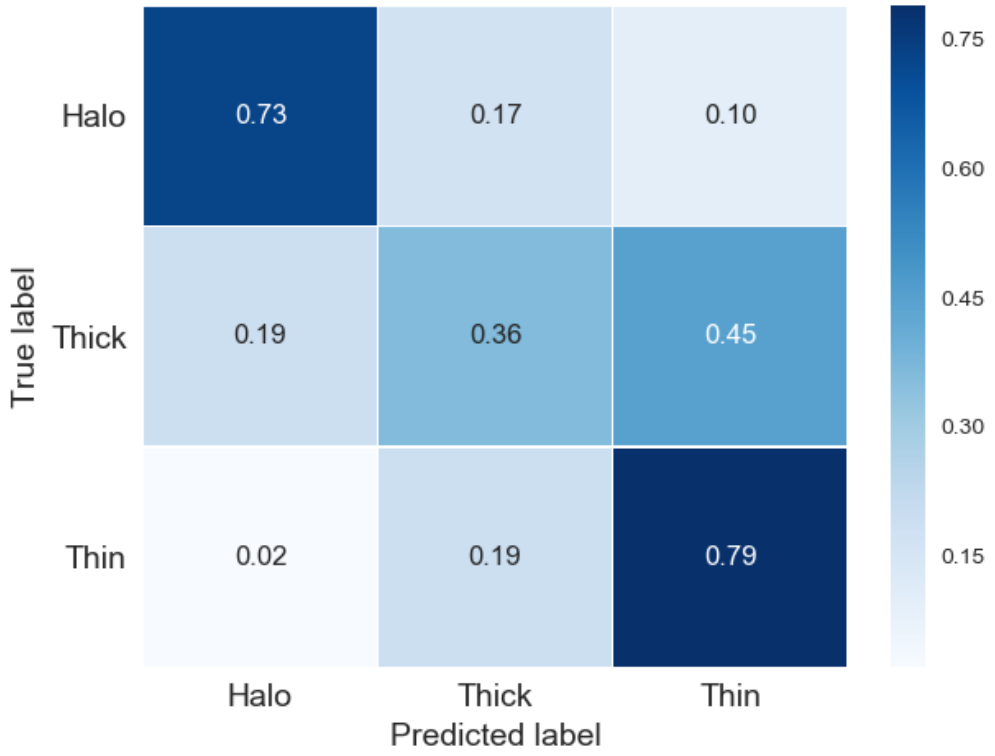

Figure 3.7: Confusion matrix of the RPM diagram applied to the test simulated sample.

Table 3.4. shows the four outcomes for the thin disk population obtained with the classical RPM diagram method. We clearly see that the amount of TP overcomes greatly the other ones. This indicates that the algorithm predicts correctly 21282 white dwarfs in the thin region over the total white dwarf population, that is a number of 26817 stars (TP + FN). Of this total amount, the classical method predicts erroneously 5138 white dwarfs in the thick region and 397 in the halo. These 5535 stars are FN. We can also see that there are 7877 white dwarfs (FP + TN) that they do not belongs to the thin class and however, the method identify 2716 white dwarfs as stars pertaining to this class. Finally, we can conclude that the prediction for the thin population is worst than using the Random Forest algorithm. Following equation 3.1 we obtain an specific accuracy of 76%.

		True label	
		Thin	Non-thin
Predicted label	Thin	21282 TP	2716 FP
	Non-thin	5535 FN	5161 TN

Table 3.4: Four outcomes of the thin population extracted from the confusion matrix using the classical method.

The corresponding result for the thick disk are presented in Table 3.5. The classical method

predicts correctly only 2016 white dwarfs (TP) out of real total of 5542 stars (TP + FN), misclassifying 3526 white dwarfs. Actually we see in the confusion matrix that 2478 stars of the thick disk are classified as thin stars and 1048 are classified as halo one. Thus, a big number of thick white dwarfs are classified erroneously like thin stars but also as halo stars. There are 29152 white dwarfs non-thick class (FP + TN) and 5534 of which are identified like thick stars. Thus we see that the number of TP is bigger than in the thin case and therefore, the prediction of correctly classified white dwarfs is really poorer. Finally, we obtain an specific accuracy of 73% for the thick disk population.

		True label	
		Thick	Non-thick
Predicted label	Thick	2016 TP	5534 FP
	Non-thick	3526 FN	23618 TN

Table 3.5: Four outcomes of the thick class extracted from the confusion matrix of the classical method.

Finally, the four outcomes of the halo population is presented in Table 3.6. Firstly, we observe a value of 1701 TP. The total real value of halo stars is 2335 (TP + FN) and the classical method identify 634 of these like non-halo white dwarfs. According to the confusion matrix as well, 396 are in the thick region and 238 are in the thin. There are 32359 white dwarfs non-halo class (FP + TN) and 1445 of which are identified within the halo population. Note that in this case, there are 30914 white dwarfs (TN) that are good non-classified. The specific accuracy, 94%. for the halo population still being very high.

		True label	
		Halo	Non-halo
Predicted label	Halo	1701 TP	1445 FP
	Non-halo	634 FN	30914 TN

Table 3.6: Four outcomes of the halo population extracted from the confusion matrix for the classical method.

In summary, doing a general analysis of the results from the classical RPM diagram method, we can infer a total accuracy of the method about 55%. If we compare this value with the accuracy of 85% of our Random Forest algorithm, it would correct to affirm that the Random Forest algorithm has resulted as a powerful tool to classify the white dwarf population.

CHAPTER 4. CLASSIFYING THE OBSERVED SAMPLE OF WHITE DWARFS. PRELIMINARY RESULTS

In previous chapters we have built, tested and improved by means of a simulated synthetic sample a Random Forest classification algorithm. Also, we have shown that the Random Forest algorithm presents a clear improvement in accuracy with respect to that of the standard method based on the RPM diagram selection. Consequently, we are now in position to apply our Random Forest algorithm to a real sample of unclassified white dwarfs. Just to remember, the white dwarf observed sample has been extracted from the IGSL sample which contains nearly 1 million stars. This pre-selection of white dwarf stars has been done by means of the RPD diagram, as shown in Figure 4.1. Gray dots represents the whole IGSL sample of stars, while red dots are our white dwarf candidates. The vertical and diagonal lines in Fig.4.1 shape the selection are of white dwarfs. The justification of these limits are the following:

- A reduced proper motion for a tangential velocity of $20\,$km/s is the minimum practical value for selecting thin disk white dwarfs avoiding contamination from main-sequence stars.
- A maximum value of $H = 30$"/yr in the reduced proper motion is the limit in accuracy and completeness that actual surveys can reach, in particular for Gaia mission.
- A minimal value of colour index $B - R = -0.5$ has been considered in order to avoid other contaminant stars rather than white dwarfs, such as dwarf stars, binary systems, among others.

After applying these selection criteria, we obtain a total number of 181300 stars that we will preliminary consider as white dwarfs stars. We remark here, that our objective is to provide a preliminary classification of this selected sample. We should take in mind that probably our observed selected sample is contaminate by other type of stars rather than white dwarfs, and a precise analysis and cross validation will be needed. However is beyond the scope of the present work to obtain such a refined sample of white dwarfs and, by no means, this possible contamination unvalidated the rigor of the selection procedure employed here.

4.1.. Distribution of the observed population

Once the Random Forest algorithm has been applied to the specific area of the observed sample, we obtained the distribution of these stars classified as thin and thick disk stars and halo stars. The Table4.1. shows the distribution obtained.

The first remarkable thing is that the vast majority of white dwarfs, a $93\,\%$, are classified as thin disk stars and only a $7\,\%$ as thick stars. Secondly, we observe the small amount of halo white dwarfs identified among the total sample. However, this result is in a reasonable agreement with other estimates for the relative proportion of stars, for instance,

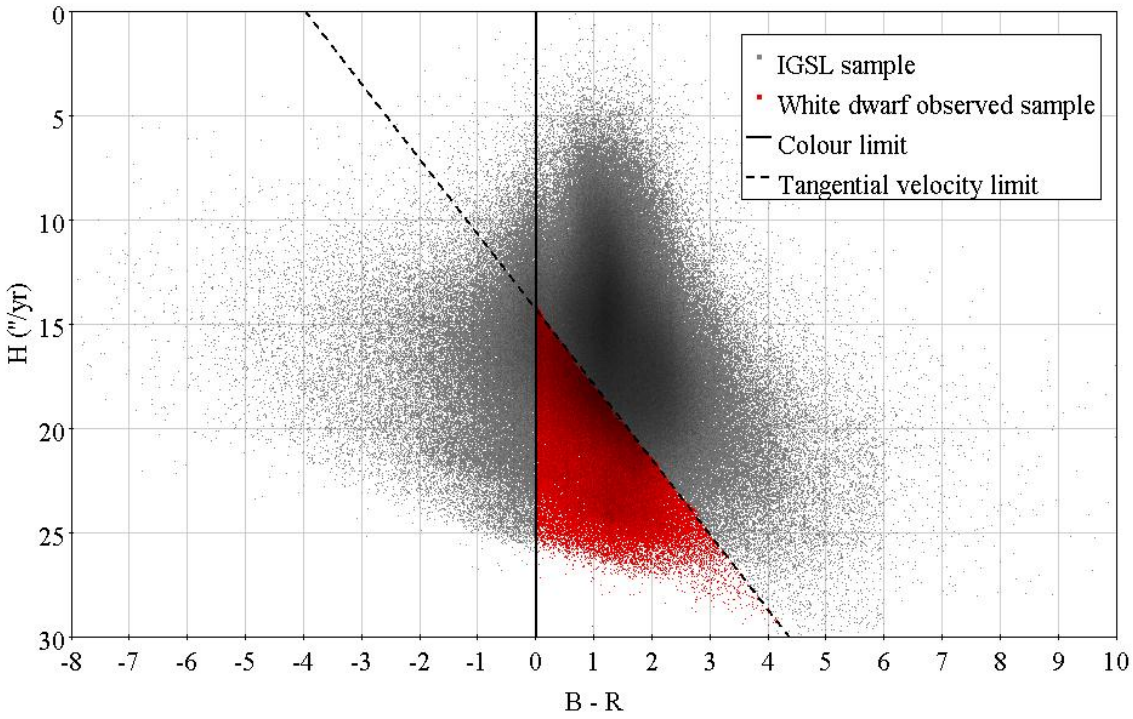

Figure 4.1: RPM diagram showing the IGSL sample (gray dots) and those objects under the area of selection (continues and dashed line), that are considered white dwarf stars (red dots).

	Halo	**Thick**	**Thin**
%	0.00	0.07	0.93
# stars	14	12587	168699

Table 4.1: Distribution of white dwarf stars into three classes after applying the Random Forest algorithm.

Fig.3.5 predicts a $79\,\%$, $16\,\%$ and $5\,\%$ for the thin, thick and halo white dwarf population, respectively.

The RPM distribution once our Random Forest algorithm has perform its classification is shown in Figure 4.2. In general terms, the obtained distribution is in accordance with what a priori is expected. However, the most noticeable discrepancy is the existence of a portion of stars, identified as thick white dwarfs, quite close to the selection limit of $V_{\mathrm{TAN}} = 20\,\mathrm{km}\cdot\mathrm{s}^{-1}$. This fact is probably related to what we discussed in a previous paragraph: the possible contamination of main-sequence stars in our selected white dwarf sample. Our Random Forest algorithm has not been training for classifying other stars than white dwarfs, so the algorithm interpret that this contaminant stars, that a minority, do not belong to the main group of stars, and consequently classified them into a secondary group. This show us that our Random Forest algorithm, is robust enough to identify *weird* objects even if it has not been trained for recognize them.

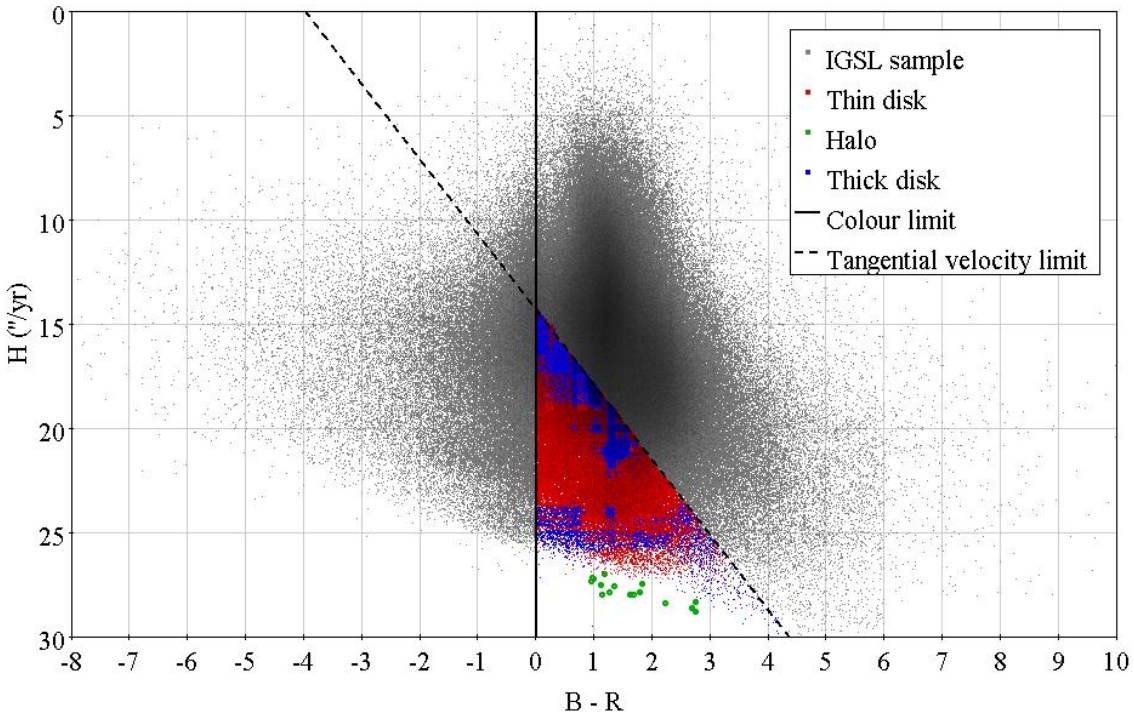

Figure 4.2: RPM diagram showing the final classification obtained by our algorithm of the thin (red dots), thick (blue dots) and halo (green dots) population.

4.2.. White dwarf luminosity functions

Luminosity function is a basic tool in the understanding of the properties of the different white dwarf populations. The luminosity function represents the number density of white dwarfs per luminosity interval. The white dwarf luminosity function carries important information about the past history, evolution and age of the Galaxy (see [13] for more information about the luminosity function). Nevertheless, our purpose here is just to obtain a preliminary luminosity function for each population.

Figure 4.3 shows the white dwarf luminosity function for the thin and thick disk population after being classified by our Random Forest algorithm. Usually, the number density is plotted against the luminosity or, equivalently the bolometric magnitude. However, given that we do not have the bolometric corrections for each star, we assume as a first approximation the absolute magnitude G as reasonable measure of the bolometric magnitude. We have disregard the halo white dwarf luminosity function, given that the number of identified halo stars is not enough for our purposes. The white dwarf luminosity function displayed fig.4.3 have been normalize to the local density in order be compared with the results from [8]. The agreement with the thin and thick luminosity function of [8] is fairly good. Our preliminary luminosity functions present a constant slope, indicative of the characteristic cooling time evolution of the white dwarfs. Besides, the slope of the thick white dwarf luminosity function is flatter than the corresponding thin population. Several reasons account for these, such a different star formation rate or the effect of a different scale height.

As a concluding remark, the luminosity functions showed in fig.4.3 in base of our Random Forest algorithm, open a door to new interesting studies in view of the expecting observation provided by future Gaia data releases.

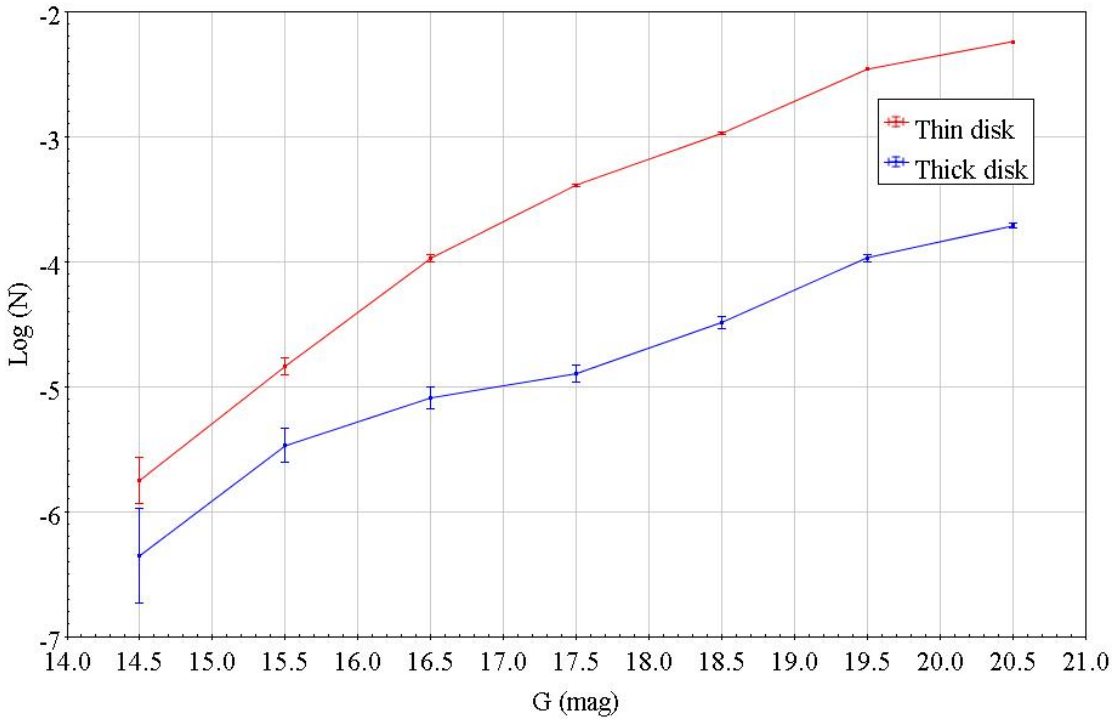

Figure 4.3: Preliminary white dwarf luminosity function for the thin and thick disk population obtained in this work.

CONCLUSIONS

It is widely known that Gaia mission will collect information about more than 1 billion stars along its five years mission lifetime. This amount of data needs to be treated with a really powerful learning and adaptation process if we are interesting to extract any specific information. In particular, since the dedicated and continuously research of the white dwarf population in the Milky Way, it was necessary a new tool to extract and classify the vast majority of these stars that the Gaia space mission will be able to detect and scan. Consequently, a new Machine Learning algorithm was purposed as the best candidate for the task. Within Machine Learning, the supervised model was selected in front of the unsupervised one.

Generally, all supervised model needs a training sample in order to fit the data and make the necessary patterns to construct the algorithm. A testing sample is also needed for visualizing the behaviour of the algorithm and be able to contrast the results with those we already know. Thus, the first chapter of this thesis has been focused on prepare the data to configure both the training and the testing sample. By means of the Initial Gaia Source List (IGSL) we have been able to harvest a large amount of stars (1 million) where white dwarf limit regime has been identified. At the same time, we built a synthetic white dwarf population provided by the Group of Astronomy and Astrophysics of the UPC. This new simulated sample of white dwarfs has been used in training and testing samples by the classification algorithms under study.

Next, different classification algorithms were considered. We analyzed five algorithms within the supervised model: Decision Tree, K-nearest neighbours, Adaboost, Random Forest and finally, Extra-Tree. We applied these algorithms to our simulated sample and then, the results were compared in order to choose the most adequate for our purpose and requirements. Summarizing, we found the following:

- Between $80\,\% - 84\,\%$ of testing accuracy is achieved from all algorithms being the Random Forest and the Extra-Tree algorithms those with more accuracy.
- Over-fitting effect appears in both the Random Forest and the Extra-Tree algorithm since the training accuracy found was $95\,\% - 100\,\%$.
- Between $80\,\% - 85\,\%$ of testing accuracy is obtained from all algorithms after applying the optimal estimators for each algorithm when trying to avoid over-fitting, being Random Forest and Extra-Tree the best ones with a maximum testing accuracy of $85\,\%$
- Random Forest algorithm takes around $16 - 20\,$s to train our training simulated sample, being the slowest of algorithms. Around $0.1 - 0.8\,$s spend the quickest algorithm, the KNN.
- The vast majority of white dwarfs are located in the thin disk of the Galaxy with percentage between $78\,\% - 85\,\%$m while thick disk counts for 0.08% - 0.14% and halo about 0.06% - 0.08%.

These results provide us a general view of how each algorithm distribute the sample. The accuracy and the time are important aspects to take into account when we decide which

algorithm is better. Both Random Forest and Extra-Tree clearly have the best accuracies but also the worst computing times (in particular in the case of Random Forest). However, we discarded Extra-Tree algorithm given that detailed analysis provide us some evidences of over-fitting problems. Although differences are really insignificant among the algorithms studied here, we considered the Random Forest algorithm as the best appropriate for this work.

The implementation of the Random Forest algorithm in the testing simulated sample, which contains a total number of 41000 white dwarfs, provided us the following main results:

- The most important features used for the algorithm to classify white dwarfs have been the proper motion and the mass with a relative importances between $0.18\,\% - 0.24\,\%$.
- Around a $85\,\%$ of specific accuracy is obtained for the thick disk population when the Randon Forest algorithm is applied, while only a $73\,\%$ is reached in the case the classical RPM diagram method.
- A $98\,\%$ of specific accuracy is obtained for halo stars for the Random Forest algorithm, while only a $94\,\%$ is reached in the case of the classical RPM diagram method.
- About $85\,\%$ total accuracy of the Random Forest algorithm is obtained in front of the $55\,\%$ for the classical method used so far.

These few statistics applied only in the simulated sample help us to understand better the behaviour of the Random Forest trained model. Although the thin population has a good performance, a resulting clearly conflict between the thin and thick disk classes are evinced by results while, on the other hand, it seems really feasible classifying halo white dwarfs. Logically, intrinsic characteristics of the thin and thick disk population are more similar than that of the halo population, consequently it seems reasonable that this conflict appears in the disk region.

Once we have totally tested the algorithm, it has been applied in a preliminary observational sample. This observational sample was extracted from IGSL observed sample (10 million stars). After applying the selection criteria we obtained a sample of white dwarfs candidates of 181300 stars. Although our analysis is quite preliminary we can affirm that our Random Forest algorithm is robust enough in order to detect possible contaminants and to discern, up to a reasonable way, among the three type population that has been trained. The algorithm also permit to identify 14 candidates to halo white dwarfs.

Although the accuracy of the Random Forest algorithm and its feasibility is quite acceptable, $85\,\%$, we consider that it is not good enough for applying to future Gaia data release. As a future work line we will briefly expose some strategies to increase it at least until a 90% of accuracy. Probably, cross validation process results as one of the easiest options to implement and specially to improve the accuracy, where a new sample is defined to ”test” the model in the training phase in order to limit problems like over-fitting. One round of cross-validation involves partitioning a sample of data into complementary subsets, performing the analysis on one subset (called the training set), and validating the analysis on the other subset (called the validation set). To reduce variability, in most methods multiple rounds of cross-validation are performed using different partitions, and the validation results are combined over the rounds to estimate a final predictive model. One of the main

reasons for using cross-validation instead of using the conventional validation (for example, in our Random Forest the total sample is divided into two sets of 70% for training which is equivalent to the 95000 white dwarfs and 30% for testing, which is equivalent to 41000 white dwarfs) is that there is not enough data available to partition it into separate training and test sets without losing significant modelling capability. In these cases, a fair way to properly estimate model prediction performance is to use cross-validation as a powerful general technique.

To sum it up, this work has presented an important challenge from the point of view of acquiring a new complex knowledge about data mining and specially, to be able to adapt it to the necessities and requirements of the thesis. Not only for programming aspects, but also for preparing the samples and controlling details. The process we are followed so far have been thought with the help of the Group of Astronomy and Astrophysics of the UPC, choosing the best way to realize it.

BIBLIOGRAPHY

[1] http://www.sdss.org/ 3

[2] https://gea.esac.esa.int/archive/ 6

[3] https://www.cosmos.esa.int/web/gaia/release 6

[4] http://www.star.bris.ac.uk/ mbt/topcat/ 7

[5] http://www.star.bris.ac.uk/ mbt/stilts/ 7

[6] R. L. Smart and L. Nicastro. *The Initial Gaia Source List*. A&A 570, A87 (2014) 7

[7] http://vizier.u-strasbg.fr/viz-bin/VizieR 7

[8] N. Rowell and N. C. Hambly. *White Dwarfs in the SuperCOSMOS Sky Survey: the Thin Disk, Thick Disk and Spheroid Luminosity Functions.* Mon. Not. R. Astron. Soc (2002). 9, 39, 45

[9] E.García-Berro, S.Torres, J.Isern and A.Burkett. *Monte Carlo simulations of the disc white dwarf population*. MNRAS 1999 vol. 302 pages 173-188. 11

[10] E.García-Berro, S.Torres, J.Isern and A.Burkett. *Monte Carlo simulations of the halo white dwarf population*. AAP 2004 vol.418 pages 53-65. 11

[11] Jos de Bruijne, Michael Perryman, Lennart Lindegren, Carme Jordi, Erik H¿g, David Katz, Mark Cropper. *Gaia astrometric, photometric, and radial-velocity performance assessment methodologies to be used by the industrial system-level teams*. Technical Note Gaia-JdB-022. June 9, 2005 12

[12] https://www.cosmos.esa.int/web/gaia/science-performance 13

[13] E.García-Berro, T.D.Oswalt. *The white dwarf luminosity function*. NAR 2016 vol. 72 pages 1-22. 45

[14] J. Isern, A. Artigas and E. García–Berro. *White dwarf cooling sequences and cosmochronology*. EPJ Web of Conferences 43, 05002 (2013). 55

[15] S. Torres, E. Garcia-Berro, J.Isern, F.Figueras. *Simulating Gaia performances on white dwarfs* Mon. Not. R. Astron. Soc. 360, 1381–1392 (2005). 56

APPENDICES

APPENDIX A. THE WHITE DWARF POPULATION

White dwarf are the final remnants of low- and intermediate- mass stars. They are the most common stellar evolutionary end-point. As a matter of fact, all stars with masses smaller than $\sim$ 10 $M_{\odot}$ will end their lives as white dwarfs. About 95% of main-sequence stars are white dwarfs and, hence, their corresponding study and analysis of provides important details about the late stages of the life of the vast majority of stars. The local population of white dwarfs carries crucial information about the physical processes that govern their own evolution as well as the structure and evolution of the Galaxy.

White dwarfs are degenerate stars, this implies that they are not able to obtain energy from nuclear burning. In other words, the material in a white dwarf no longer undergoes fusion reactions, so the star has no source of energy. As a result, it cannot support itself by the heat generated by fusion against gravitational collapse, but is supported only by electron degeneracy pressure, causing it to be extremely dense. A white dwarf is very hot when it forms, but given that it has no source of energy, it will gradually radiate its energy and cool down. This implies that its luminosity, which initially has a hot colour temperature, will cool and redden with time. After a very long time, a white dwarf will cool down and its material will begin to crystallize. Theoretically, white dwarfs low temperature means it will no longer emit significant heat or light, and it will become a cold *black dwarf*. This process is so much slow that it is impossible to see a those objects within the actual Universe age. For more detailed information about the formation and evolution of white dwarfs, see [14].

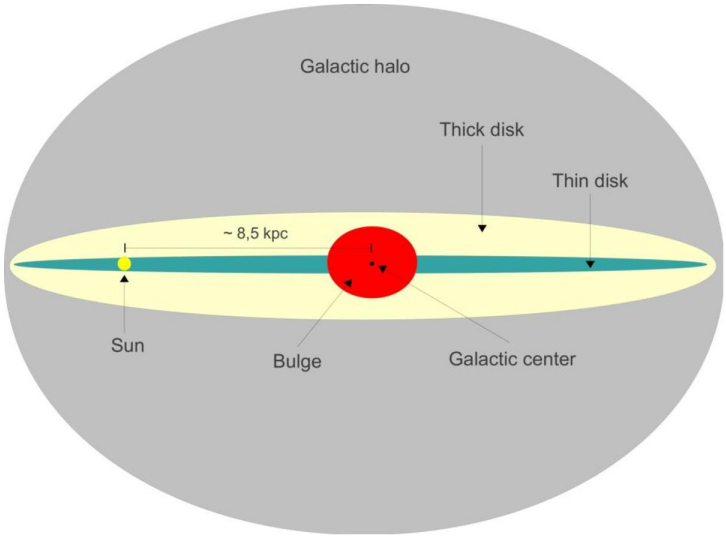

Figure A.1: Components of the Galaxy: thin, thick and halo.

These kind of stars can be classified according to their location in the Galaxy. For instance, our interesting region are: galactic halo or spheroid, thin disk and thick disk (Fig. A.1). It is well-known the local number of white dwarfs in the disk of the Galaxy overcomes the case of halo star. On the other hand, the halo is believed to be the oldest component of the Galaxy. As such, halo stars are generally older an cooler than disk stars. Disk stars (among them our Sun) have a tightly constrained distribution of velocities given that they corotate in the Galactic plane, while halo stars generally have very high velocities given that is distribution of velocities are randomly distributed in spherical symmetric profile with respect the Galactic center.

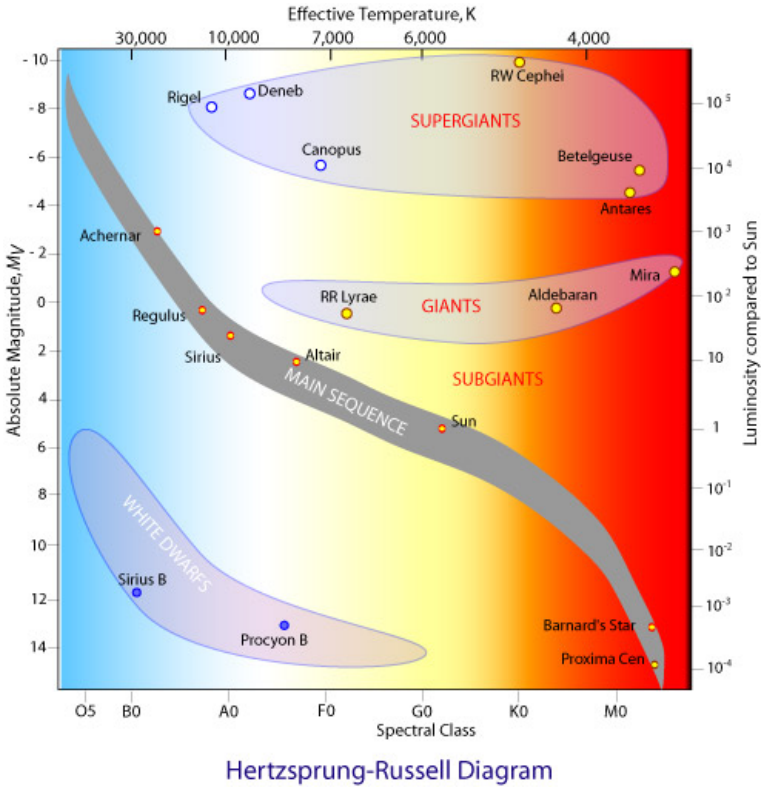

Figure A.2: Hertzsprung-Russell diagram showing the location of the main type of stars. White dwarfs locates in the left bottom corner of the diagram under the main-sequence track.

On the other hand, white dwarfs and other kind of stars can be distinguished to the rest by means of the renowned colour- magnitude diagram as a variation of the Herzstprung-Russell diagram (HR) in which the effective temperature against the luminosity is plotted. This variation, equal to the original HR diagram, shows a group of stars in various stages of their evolution in terms of luminosity, colour, effective temperature or luminosity physical parameters. As Fig.A.2 shows, main sequence stars vary widely in effective temperature but the hotter they are, the more luminous they are, hence the main sequence tends to follow a track starting from the bottom right of the diagram to the top left. These stars are fusing hydrogen into helium in their cores. Stars spend the bulk of their existence as main sequence stars. Other major groups of stars found on the H-R diagram are the giants and supergiants; luminous stars that have evolved off the main sequence, and finally our interesting case, white dwarfs. Therefore, by means of such a diagrams we are able to classify the stars according to their physical properties.

For more information about the white dwarfs distribution and classification on the Galaxy see [15].

APPENDIX B. SQL SCRIPT TO TOPCAT

In this appendix the ADQL/SQL script to extract the Observed sample is showed. It has been applied to the TAP-query of the TOPCAT (Fig.1.2) as new commands. We observe that we take the desired parameters as well as to put some conditions in the data, for example G magnitude must be lower than 21 mag (Gaia photometric limit).

$$
\begin{aligned}
&SELECTALL \\
&dec, \\
&dec_error, \\
&ra, \\
&ra_error, \\
&mag_bj, \\
&mag_bj_error, \\
&mag_rf, \\
&mag_rf_error, \\
&mag_g, \\
&mag_error, \\
&mag_bj - mag_rfASBV, \\
&pm_dec, \\
&pm_ra, \\
&SQRT(POWER(pm_ra*POWER(10,-3),2)+POWER(pm_dec*POWER(10,-3),2))ASpm, mag_bj+ \\
&5*log10(SQRT(POWER(pm_ra*POWER(10,-3),2)+POWER(pm_dec*POWER(10,-3),2)))+ \\
&5ASHbj, \\
&mag_bj+5*log10(20)-3.38ASHbj_vtan \\
&FROMpublic.igsl_source \\
&WHERESQRT(POWER(pm_ra,2)+POWER(pm_dec,2))>0ANDmag_g<21
\end{aligned}
$$

APPENDIX C. THE RECEIVER OPERATING CHARACTERISTIC CURVE

First of all, we are going to explain some important notions of sensitivity and specificity of a test to explain the Receiver Operating Characteristics curve (ROC curve hereafter). The sensitivity is defined as the probability of the prediction rule or model predicting an observation as 'positive' given that is truth. In other words, the sensitivity is the proportion of truly positive observations which is classified as such by the model. Conversely, the specificity is the probability of the model predicting 'negative' given that the observation is 'negative'.

Our model is perfect at classifying observations if it has 100% sensitivity and 100% specificity. Unfortunately in practice this is not attainable. Therefore, how can we summarize the discrimination ability of our model? For each observation, our fitted model can be used to calculate the fitted probabilities. On their own, these do not tell us how to classify observations as positive or negative. One way to create such a classification rule is to choose a cut-point (cp), and classify those observations with a fitted probability above this cp as positive and those at or below it as negative. For this particular cut-off, we can estimate the sensitivity by the proportion of observations with which have a predicted probability above cp, and similarly we can estimate specificity by the proportion of observations with a predicted probability at or below cp. If we increase the cut-point cp, fewer observations will be predicted as positive. This will mean that fewer of the observations will be predicted as positive (reduced sensitivity), but more of the observations will be predicted as negative (increased specificity).

Now we come to the ROC curve showing a general example of it in the figure C.1, which is simply a plot of the values of sensitivity against one minus specificity, as the value of the cut-point cp is increased from 0 through to 1.

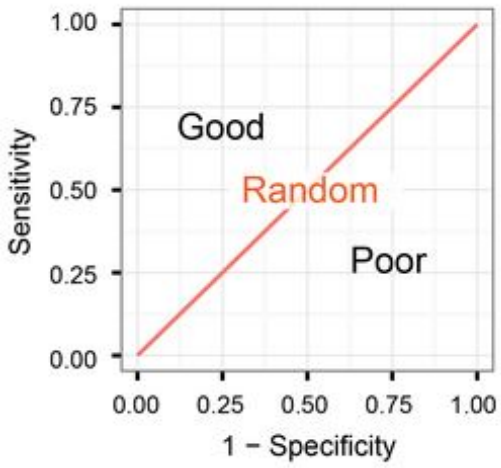

Figure C.1: General configuration of a ROC curve.

A model with high discrimination ability will have high sensitivity and specificity simultaneously, leading to an ROC curve which goes close to the top left corner of the plot (ideal curve). A model with no discrimination ability will have an ROC curve which is the 45 degree diagonal line (the worst prediction in a model). Therefore and following the repre-

sentation of the Fig.C.1, while ROC curves of each label are above the diagonal line (called Random in the plot), our prediction will be positive, good or excellent, always depending on how near it is of the top left corner. This can be also seen in terms of the area of each curve. The larger area below the line, the better the prediction is.

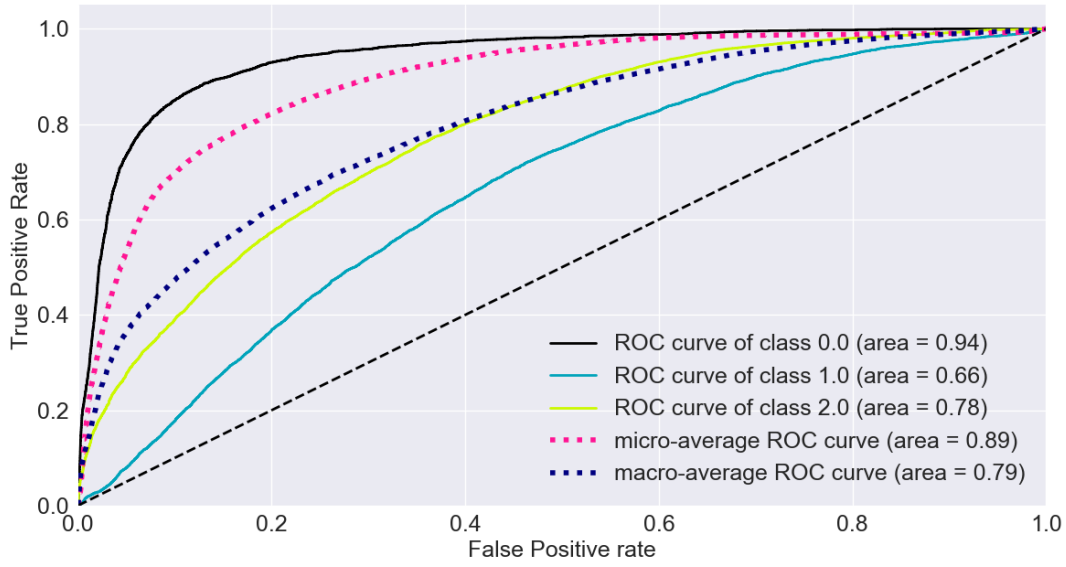

Figure C.2: ROC curve of the Random Forest algorithm applied to the test simulated sample. Class 0 is the Halo label, class 1 is the Thick disk label and finally, class 2 is the Thin disk label.

In Figure C.2, the curve closer to the top left corner and therefore with more area below itself is the Halo class (class 0 in the plot) with a 94% of the area. The next one would be the Thin class (class 2) with a 78% of the area, and lastly the Thick class (class 1) with only the 66% of the area. Therefore, we clearly see in Fig.C.2 that the better prediction is for the Halo white dwarfs and the worst for thick ones. This result justify the different distributions of the four outcomes from the confusion matrix where there were few halo white dwarfs but most of them were focused on the diagonal of the matrix. The ROC curve in Fig.C.2 remarks that our Random Forest algorithm has a little bit of conflict to distinguish between white dwarfs from the thin disk and from the thick disk, being thick white dwarfs the more adversely affect in front of the thin ones. We repeat again that this probably is caused by the residual over-fitting effect in the sample.

As a summary, in view of the ROC curve presented in C.2 we could say that the halo classification is excellent, the thin classification is good and the thick classification is a little bit poorer. In any case, we must pay attention to the mean rates lines where the mean model behaviour is reflected, and where we obtained really good areas.

APPENDIX D. THE CODE

In this appendix the complete code of the new Machine Learning algorithm and its specific modifications is showed. Once the code has been optimized to the final version, it contains 1050 coding lines. Basically, the code is structured by three principal parts:

- Libraries: Imports from all libraries needed to call those functions to realize some actions such as train the data, between others.
- Methods and functions: The entire program is configured through different methods which realize different actions. By this way, the code has a certain order an it is easier to work with it.
- The main: This method is where the program starts to run and is where all the previous commented methods are called.

Specifically, within methods part we can find methods for data treatment, for algorithm configuration, for different statistics and finally for plotting results. Since we are working with Python, we have taken advantage of storing the data by means of dictionaries, with which both the simulated sample and the observed sample are dictionaries. This is done by this way for a better operation and organization in the code. All files scanned and created from the program have the CSV format.

Thus, from next page you will find the complete code used for this work.

```
#TITLE:    IMPLEMENTATION OF A RANDOM FOREST MACHINE LEARNING ALGORITHM IN THE CONTEXT
#                                  GAIA SPACE MISSION

#AUTHOR OF THE CODE:               CARLES CANTERO MITJANS

#TUTOR:                            DR. SANTIAGO TORRES

#INSTITUTION:             UNIVERSITAT POLITÈCNICA DE CATALUNYA (UPC)

#              ESCOLA D'ENGINYERIA DE TELECOMUNICACIONS Y AEROSPACIAL DE CASTELLDEFELS

#FINAL DATE:                         FEBRUARY, 8  2018


#------------------------------------------------------------------------------
###############################################################################
##############################  LIBRARIES  ####################################
###############################################################################
#------------------------------------------------------------------------------

## FROM HERE ALL USED LIBRARIES FOR PLOTTING AND COMPUTING
from matplotlib.pylab import *
import csv
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns;
import graphviz
import time
from sklearn import tree
sns.set()
from collections import OrderedDict
import scikitplot as skplt

#SKLEARN ALGORITHM LIBRARIES
from sklearn.model_selection import train_test_split
from sklearn.ensemble import RandomForestClassifier
from sklearn.tree import  DecisionTreeClassifier
from sklearn.neighbors import KNeighborsClassifier
from sklearn.ensemble import ExtraTreesClassifier
from sklearn.ensemble import AdaBoostClassifier
from sklearn.datasets import make_classification
from sklearn.ensemble import RandomForestClassifier as RFC
from sklearn.externals import joblib
from sklearn.naive_bayes import GaussianNB


#------------------------------------------------------------------------------
###############################################################################
####################  METHODS AND FUNCTIONS  ##################################
###############################################################################
#------------------------------------------------------------------------------

## ..............................DATA METHODS..............................

def Read_csv(path):

    #...This method returns the data readed from a CSV file..
```

```
 data = pd.read_csv(path)
    return data

def Guardar_modelo(X_train, Y_train):

    # ...This method train and save the Random Forest model in a PKL file...

    clf = RandomForestClassifier(n estimators=51, max depth=16)
    clf.fit(X train, Y train)
    joblib.dump(clf, "modelo_entrenado_1.pkl")



def Importar_modelo(X_train, Y_train):

    #...This method imports and returns the saved PKL file of the trained and saved
method...

    clf =
joblib.load("C:/Users/kanti/PycharmProjects/Gaia_Machine_Learning/modelo_entrenado_1.pkl
")
    return clf

def Extract_data_RPMD (X_train, Y_train, X_test):

    #...This method is dedicated to extract those predicted datafrom simulated sample
for plotting the new RPM diagram...

    #Declaration of matrices for computation the predicted results
    x halo, y halo = [], []
    x thick, y thick = [], []
    x thin, y thin = [], []

    # Training the Random Forest with the optimal parameters
    clf = RandomForestClassifier(n_estimators=51, max_depth=16)
    clf.fit(X train, Y train)
    results = clf.predict(X_test) # predicted labels

    #Choosing the parameters H and (B-R) from the test sample
    H = [row[7] for row in X_test]
    BR = [row[4] for row in X_test]

    #Becoming the selected data (H, B-R and predicted labels) to arrays for a better
operation
    Hnew = np.array(H)
    BRnew = np.array(BR)
    results_new = np.array(results)

    #Building a global matrix with all parmeters together
    matrix = np.column_stack((Hnew, BRnew))
    matrix_new = np.column_stack((matrix, results_new))

    #This loop is in charge to run our new matrix and it distinguish those stars from
halo(0), thick (1) and thin(2)
    j = 0
    for i in matrix_new:

        # Halo conditional
        if matrix_new[j][2] == 0:
            print("halo")
            x_halo.append(matrix_new[j][1])
            y_halo.append(matrix_new[j][0])

        #Thick disk conditional
        elif matrix_new[j][2] == 1:
            print("thick")
            x_thick.append(matrix_new[j][1])
            y_thick.append(matrix_new[j][0])

        #Thin disk conditional
        elif matrix_new[j][2] == 2:
            print("thin")
            x_thin.append(matrix_new[j][1])
            y_thin.append(matrix_new[j][0])

        j = j + 1
```

```
    #Putting together those counters from each class into new single variables
    data halo = np.column stack((x halo, y halo))
    data thick = np.column stack((x thick, y thick))
    data thin = np.column stack((x thin, y thin))

    #Making a new file with the selected data from each class
    with open("triangulo_halo.csv", "w") as f:
        writer = csv.writer(f)
        writer.writerows(data halo) #(data halo or data thick or data thin)

def Extract_data_histogram (X_train, Y_train, X_test):

    # ...This method extract those configured data for building a new histogram...

    #Calling the method "predict RandomForest Observed"
    results = predict RandomForest Observed(X test, X train, Y train)
    results_new = np.array(results)

    # Choosing the parameter G from the test sample and becoming it into an array
    G = [row[2] for row in X_test]
    G new = np.array(G)

    # Putting together those counters from each class into new single variables
    matrix = np.column_stack((G_new, results_new))

    #All needed counters for the histogram computation
    g1, g2, g3, g4, g5 = 0, 0, 0, 0, 0
    g6, g7, g8, g9, g10 = 0, 0, 0, 0, 0
    g11, g12, g13, g14, g15, g16, g17, g18,g19 = 0, 0, 0, 0, 0, 0, 0, 0,0
    gg = 0
    j = 0

    #Loop to build the histogram with a range in G magnitude between 7 and 27
magnitudes.
    for i in matrix:
        if matrix[j][1] == 2: #here we change the label (0,1 or 2)

            if 7 < matrix[j][0] <= 8:
                gg = gg + 1
            if 8 < matrix[j][0] <= 9:
                g1 = g1 + 1
            if 9 < matrix[j][0] <= 10:
                g2 = g2 + 1
            if 10 < matrix[j][0] <= 11:
                g3 = g3 + 1
            if 11 < matrix[j][0] <= 12:
                g4 = g4 + 1
            if 12 < matrix[j][0] <= 13:
                g5 = g5 + 1
            if 13 < matrix[j][0] <= 14:
                g6 = g6 + 1
            if 14 < matrix[j][0] <= 15:
                g7 = g7 + 1
            if 15 < matrix[j][0] <= 16:
                g8 = g8 + 1
            if 16 < matrix[j][0] <= 17:
                g9 = g9 + 1
            if 17 < matrix[j][0] <= 18:
                g10 = g10 + 1
            if 18 < matrix[j][0] <= 19:
                g11 = g11 + 1
            if 19 < matrix[j][0] <= 20:
                g12 = g12 + 1
            if 20 < matrix[j][0] <= 21:
                g13 = g13 + 1
            if 21 < matrix[j][0] <= 22:
                g14 = g14 + 1
            if 22 < matrix[j][0] <= 23:
                g15 = g15 + 1
            if 23 < matrix[j][0] <= 24:
                g16 = g16 + 1
            if 24 < matrix[j][0] <= 25:
                g17 = g17 + 1
            if 25 < matrix[j][0] <= 26:
                g18 = g18 + 1
            if 26 < matrix[j][0] <= 27:
                g19 = g19 + 1
```

```
        j = j + 1


    # We compute the errors of each measure
    error_gg = sqrt(gg)
    error g1 = sqrt(g1)
    error g2 = sqrt(g2)
    error g3 = sqrt(g3)
    error g4 = sqrt(g4)
    error_g5 = sqrt(g5)
    error_g6 = sqrt(g6)
    error g7 = sqrt(g7)
    error g8 = sqrt(g8)
    error g9 = sqrt(g9)
    error g10 = sqrt(g10)
    error_g11 = sqrt(g11)
    error_g12 = sqrt(g12)
    error g13 = sqrt(g13)
    error g14 = sqrt(g14)

    # We compute the the logarithm propagation of each error
    errorLog_gg = error_gg / gg
    errorLog_g1 = error_g1 / g1
    errorLog g2 = error g2/g2
    errorLog g3 = error g3/g3
    errorLog g4 = error g4/g4
    errorLog_g5 = error_g5/g5
    errorLog_g6 = error_g6/g6
    errorLog g7 = error g7/g7
    errorLog g8 = error g8/g8
    errorLog g9 = error g9/g9
    errorLog_g10 = error_g10/g10
    errorLog_g11 = error_g11/g11
    errorLog_g12 = error_g12/g12
    errorLog g13 = error g13/g13
    errorLog g14 = error g14 / g14

    #Re-scalling the log(N) according to the other article results
    rescale = 7.2
    ggn = math.log10(gg) - rescale
    g1n = math.log10(g1) - rescale
    g2n = math.log10(g2) - rescale
    g3n = math.log10(g3) - rescale
    g4n = math.log10(g4) - rescale
    g5n = math.log10(g5) - rescale
    g6n = math.log10(g6) - rescale
    g7n = math.log10(g7) - rescale
    g8n = math.log10(g8) - rescale
    g9n = math.log10(g9) - rescale
    g10n = math.log10(g10) - rescale
    g11n = math.log10(g11) - rescale
    g12n = math.log10(g12) - rescale
    g13n = math.log10(g13) - rescale
    g14n = math.log10(g14) - rescale

    #Putting together all variables needed for th histogram
    log_numbers = np.array([g7n, g8n, g9n, g10n, g11n,g12n,g13n])
    values = np.array([14.5, 15.5, 16.5, 17.5, 18.5, 19.5, 20.5])
    errors = np.array([errorLog g7, errorLog g8,errorLog g9, errorLog g10,
errorLog_g11,errorLog_g12,errorLog_g13])
    data_halo = np.column_stack((values, log_numbers))
    data complete = np.column stack((data halo,errors))

    # Making a new CSV file with the selected data from each class
    with open("Lumi_thin.csv", "w") as f:
        writer = csv.writer(f)
        writer.writerows(data_complete)


## ........................... ALGORITHMS METHODS ............................

def RandomForest_Classifier(X_train, Y_train):

    #...This method is in charge to train the data through the RANDOM FOREST
ALGORITHM...
```

```
    clf = RandomForestClassifier(n estimators=51, max depth=16)
    clf.fit(X train, Y train)
    return clf


def predict_RandomForest(X_test, X_train, Y_train):

    # ...This method is in charge to predict the test data through the RF trained
model...

    clf = RandomForest_Classifier(X_train,Y_train)
    results = clf.predict(X test)
    return results

def predict_RandomForest_Observed(Observed_test, X_train, Y_train)
    # ... This peculiar method serve us to apply the saved model of RF and also to
predict...

    X test = pd.DataFrame(Observed test)
    X_test.fillna(X_test.mean(), inplace=True)
    clf = Importar_modelo(X_train, Y_train)
    results = clf.predict(X_test)
    return results


def KNN_Classifier(X_train,Y_train):

    # ...This method is in charge to train the data through the KNN ALGORITHM...

    clf = KNeighborsClassifier(n neighbors= 30)
    clf.fit(X train,Y train)
    return clf

def predict_KNN(X_train,Y_train,X_test):

    # ...This method is in charge to predict the test data through the KNN trained
model...

    results = KNN_Classifier(X_train,Y_train).predict(X_test)
    return results



def DecisionTree_Classifier(X_train, Y_train):

    # ...This method is in charge to train the data through the DECISION TREE
ALGORITHM...

    clf = DecisionTreeClassifier(max depth=8)
    clf.fit(X_train,Y_train)
    return clf


def predict_DecisionTree(X_train,Y_train,X_test):

    # ...This method is in charge to predict the test data through the DT trained
model...

    results = DecisionTree Classifier(X train,Y train).predict(X test)
    return results

def Extratrees_Classifier(X_train,Y_train):

    # ...This method is in charge to train the data through the EXTRA TREE ALGORITHM...

    clf = ExtraTreesClassifier(n estimators=51, max depth=24)
    clf.fit(X_train,Y_train)
    return clf

def predict_Extratrees(X_train,Y_train,X_test):

    # ...This method is in charge to predict the test data through the DT trained
model...

    results = Extratrees_Classifier(X_train,Y_train).predict(X_test)
```

```
    return results


def Adaboost_Classifier(X_train,Y_train):

    # ...This method is in charge to train the data through the ADABOOST ALGORITHM...

    clf = AdaBoostClassifier(n estimators=70)
    clf.fit(X train,Y train)
    return clf


def predict_Adaboost(X_train,Y_train,X_test):

    # ...This method is in charge to predict the test data through the AB trained
model...

    results = Adaboost Classifier(X train,Y train).predict(X test)
    return results


## ......................... STATISTICS ..............................

def recuento(X_train, Y_train, Observed_test):

    # ... This method  realize an statistics about how many WD's are in each predicted
class...

    #Calling the appropiate method to predict with Random Forest
    results = predict RandomForest Observed(Observed test, X train, Y train)

    #Counters declared
    halo, thin, thick, compt = 0, 0, 0, 0

    #Loop to count stars in each class
    for i in results:
        compt = compt + 1
        if i == 2:
            thin = thin + 1
        elif i == 1:
            thick = thick + 1
        elif i == 0:
            halo = halo + 1

    #We compute the percentage of each class
    estHalo = (halo/ compt)*100
    estThick = (thick / compt) * 100
    estThin = (thin / compt) * 100

    #Showing the results
    print("# halo: ",halo,"(",estHalo,"%)")
    print("# thick: ", thick,"(",estThick,"%)")
    print("# thin: ", thin,"(",estThin,"%)")
    print("Total:", compt)


def RandomForest_Score(X_train,Y_train,X_test,Y_test):

    # ...This method uses Random Forest to compute a general score of the model...

    #Train score
    score train = RandomForest Classifier(X train, Y train).score(X train, Y train)

    #Test score
    score_test = RandomForest_Classifier(X_train,Y_train).score(X_test,Y_test)

    #Errors computation
    error train = 1 - score train
    error test = 1 - score test

    #Showing the results
    print("Score train:", score_train)
    print("Score test:", score_test )
    print("Error train:", error_train)
    print("Error test:", error_test)
```

```
    return {"str":score_train, "ste":score_test, "etr":error_train, "ete":error_test}


def KNN_Score(X_train,Y_train,X_test,Y_test):

    # ...This method uses KNN to compute a general score of the model...

    #Train score
    score train = KNN Classifier(X train,Y train).score(X train,Y train)

    #Test score
    score test = KNN Classifier(X train,Y train).score(X test,Y test)

    # Errors computation
    error train = 1 - score train
    error_test = 1 - score_test

    # Showing the results
    print("Score train:", score_train)
    print("Score test:", score_test )
    print("Error train:", error_train)
    print("Error test:", error_test)

    return score_train, score_test, error_train, error_test


def DecisionTree_Score(X_train,Y_train,X_test,Y_test):

    # ...This method uses Decision Tree to compute a general score of the model...


    #Train score
    score_train = DecisionTree_Classifier(X_train,Y_train).score(X_train,Y_train)

    #Test score
    score test = DecisionTree Classifier(X train,Y train).score(X test,Y test)

    # Errors computation
    error_train = 1 - score_train
    error_test = 1 - score_test

    # Showing the results
    print("Score train:", score_train)
    print("Score test:", score_test )
    print("Error train:", error_train)
    print("Error test:", error_test)

    return  score_train, score_test, error_train, error_test



def Extratrees_Score(X_train,Y_train,X_test,Y_test):

    # ...This method uses Extra Tree to compute a general score of the model...

    #Train score
    score_train = Extratrees_Classifier(X_train,Y_train).score(X_train,Y_train)

    #Test score
    score test = Extratrees Classifier(X train,Y train).score(X test, Y test)

    # Errors computation
    error train = 1 - score train
    error test = 1 - score test

    # Showing the results
    print("Score train:", score_train)
    print("Score test:", score_test )
    print("Error train:", error_train)
    print("Error test:", error_test)

    return score_train, score_test, error_train, error_test


def Adaboost_Score(X_train,Y_train,X_test,Y_test):
```

```
    # ...This method uses Adaboost to compute a general score of the model...

    #Train score
    score_train = Adaboost_Classifier(X_train,Y_train).score(X_train,Y_train)

    #Test score
    score test = Adaboost Classifier(X train,Y train).score(X test,Y test)

    # Errors computation
    error_train = 1 - score_train
    error_test = 1 - score_test

    # Showing the results
    print("Score train:", score_train)
    print("Score test:", score_test )
    print("Error train:", error_train)
    print("Error test:", error_test)

    return score_train, score_test, error_train, error_test


## ........................... PLOTS ..................................

def plot_confusion_matrix(cm, classes,normalize=False, cmap=plt.cm.Blues):

    #...This function prints and plots the confusion matrix. Normalization can be
applied by setting `normalize=True ...

    #Conditional for normalize the CM
    if normalize:
        cm = cm.astype('float') / cm.sum(axis=1)[:, np.newaxis]
        print("Normalized confusion matrix")
    else:
        print('Confusion matrix, without normalization')

    #We use the imshow mode for creating the background of the CM
    plt.imshow(cm, interpolation='nearest', cmap=cmap)

    plt.colorbar()#Plotting the color bar

    #Modifying tick marks
    tick marks = np.arange(len(classes))
    plt.xticks(tick_marks, classes, rotation=45, fontsize = 12)
    plt.yticks(tick_marks, classes, fontsize=12)

    #Loop for the contrast of the colour in the map of the CM
    fmt = '.2f' if normalize else 'd'
    thresh = cm.max() / 2.
    for i, j in itertools.product(range(cm.shape[0]), range(cm.shape[1])):
        plt.text(j, i, format(cm[i, j], fmt),
                 horizontalalignment="center",
                 color="white" if cm[i, j] > thresh else "black")

    #We plot the CM
    plt.tight_layout()
    plt.ylabel('True label', fontsize=12)
    plt.xlabel('Predicted label', fontsize=12)
    plt.rc("xtick", labelsize=14)
    plt.rc("ytick", labelsize=14)
    plt.show()



def Plot_Tree(X_train, Y_train,f_names,t_names):

    # ...This particular method uses "graphviz" which help us to visualize one decision
tree with its configuration...
    clf = DecisionTree_Classifier(X_train,Y_train)

    dot_data = tree.export_graphviz(clf, out_file=None,
                                    feature names=f names,
                                    class names=t names,
                                    filled=True, rounded=True,
                                    special_characters=True)
    graph = graphviz.Source(dot_data)
```

```
    graph.render("tree")



def Plot_ROC_curves(X_train,Y_train, X_test, Y_test):

    #...This method plot the ROC curve by means of the GaussianNB function...

    nb = GaussianNB()
    nb = nb.fit(X train, Y train)
    y probas = nb.predict proba(X test)
    skplt.metrics.plot_roc_curve(Y_test, y_probas, title="", text_fontsize=18)
    plt.ylabel("True Positive Rate",fontsize=18)
    plt.xlabel("False Positive rate", fontsize=18)
    plt.show()

def Plot_accuracy_estimators(X_train,Y_train,X_test,Y_test):


    # ...This method plot the variation of the accuracy of each method in front of their
own estimators...


    #Counters
    i, compt = 1, 1

    #Arrays
    RF x axis, RF y axis = [], []
    ET_x_axis, ET_y_axis = [], []
    AB_x_axis, AB_y_axis = [], []
    KNN x axis, KNN y axis = [], []
    KNN y axisTrain, RF y axisTrain, ET y axisTrain, AB y axisTrain = [], [], [], []

    # Loop that for each method (compt) compute the accuracy varying the estimator (i)
    iterations = 60
    while compt <= 4 :
        if compt == 1: #Random Forest
            RF clf = RandomForestClassifier(n estimators=i, max depth=17)
            RF clf.fit(X train, Y train)
            RF_score_train = RF_clf.score(X_train, Y_train)
            RF_score_test = RF_clf.score(X_test, Y_test)
            RF_y_axis.append(RF_score_test)
            RF y axisTrain.append(RF score train)
            RF x axis.append(i)

        elif compt == 2: #Extra Tree
            ET_clf = ExtraTreesClassifier(n_estimators=i,max_depth=26)
            ET_clf.fit(X_train, Y_train)
            ET score train = ET clf.score(X train, Y train)
            ET score test = ET clf.score(X test, Y test)
            ET_y_axis.append(ET_score_test)
            ET_y_axisTrain.append(ET_score_train)
            ET_x_axis.append(i)

        elif compt == 3: #Adaboost
            AB_clf = AdaBoostClassifier(n_estimators=i)
            AB_clf.fit(X_train, Y_train)
            AB_score_train = AB_clf.score(X_train, Y_train)
            AB_score_test = AB_clf.score(X_test, Y_test)
            AB y axis.append(AB score test)
            AB y axisTrain.append(AB score train)
            AB_x_axis.append(i)

        elif compt == 4: #KNN
            KNN clf = KNeighborsClassifier(n neighbors=i)
            KNN clf.fit(X train, Y train)
            KNN score train = KNN clf.score(X train, Y train)
            KNN_score_test = KNN_clf.score(X_test, Y_test)
            KNN_y_axis.append(KNN_score_test)
            KNN y axisTrain.append(KNN score train)
            KNN x axis.append(i)

        i = i + 1

        if i == iterations:
            i = 1
```

```
            compt = compt + 1

    #Plotting
    plt.rc("xtick", labelsize=18)
    plt.rc("ytick", labelsize=18)
    plt.plot(RF_x_axis, RF_y_axis, c="r", label="Random Forest test line")
    plt.plot(RF_x_axis, RF_y_axisTrain, c="r", linestyle="--", label="Random Forest
train line")
    plt.plot(ET_x_axis, ET_y_axis, c="b", label="Extra-trees test line")
    plt.plot(ET_x_axis, ET_y_axisTrain, c="b", linestyle="--", label="Extra-trees train
line")
    plt.plot(AB_x_axis, AB_y_axis, c="g", label="Adaboost test line")
    plt.plot(AB_x_axis, AB_y_axisTrain, c="g", linestyle="--", label="Adaboost train
line")
    plt.plot(KNN_x_axis, KNN_y_axis, c="c", label="KNN test line")
    plt.plot(KNN_x_axis, KNN_y_axisTrain, c="c", linestyle="--", label="KNN train line")
    plt.xlabel('# Estimators', fontsize=18)
    plt.ylabel('Accuracy (%)', fontsize=18)
    plt.axis([0,60,0.65,1.01])
    plt.legend(loc=0,fontsize= 14)
    plt.show()

def Plot_accuracy_maxDepth(X_train,Y_train,X_test,Y_test):

    # ...This method plot the variation of the maximum depth of each method in front of
their own estimators...

    #Counters
    i, compt = 1, 1

    #Arrays
    RF x axis, RF y axis = [], []
    ET_x_axis, ET_y_axis = [], []
    DT_x_axis, DT_y_axis = [], []
    DT_y_axisTrain, RF_y_axisTrain, ET_y_axisTrain, AB_y_axisTrain = [], [], [], []

    # Loop that for each method (compt) compute the maximum Depth varying the estimator
(i)
    iterations = 50
    while compt <= 3 :
        if compt == 1: #Random Forest
            RF clf = RandomForestClassifier(max depth=i)
            RF clf.fit(X train, Y train)
            RF_score_train = RF_clf.score(X_train, Y_train)
            RF_score_test = RF_clf.score(X_test, Y_test)
            RF_y_axis.append(RF_score_test)
            RF_y_axisTrain.append(RF_score_train)
            RF x axis.append(i)

        elif compt == 2: #Extra Tree
            ET_clf = ExtraTreesClassifier(max_depth=i)
            ET_clf.fit(X_train, Y_train)
            ET score train = ET clf.score(X train, Y train)
            ET score test = ET clf.score(X test, Y test)
            ET_y_axis.append(ET_score_test)
            ET_y_axisTrain.append(ET_score_train)
            ET_x_axis.append(i)

        elif compt == 3: #Decision Tree
            DT clf = DecisionTreeClassifier(max depth=i)
            DT_clf.fit(X_train, Y_train)
            DT_score_train = DT_clf.score(X_train, Y_train)
            DT score test = DT clf.score(X test, Y test)
            DT y axis.append(DT score test)
            DT y axisTrain.append(DT score train)
            DT x axis.append(i)


        i = i + 1

        if i == iterations:
            i = 1
            compt = compt + 1

    #Plotting the figure
    plt.rc("xtick", labelsize=18)
    plt.rc("ytick", labelsize=18)
```

```
    plt.plot(RF_x_axis, RF_y_axis, c="r", label="Random Forest test line")
    plt.plot(RF_x_axis, RF_y_axisTrain, c="r", linestyle="--", label="Random Forest
train line")
    plt.plot(ET_x_axis, ET_y_axis, c="b", label="Extra-trees test line")
    plt.plot(ET_x_axis, ET_y_axisTrain, c="b", linestyle="--", label="Extra-trees train
line")
    plt.plot(DT_x_axis, DT_y_axis, c="g", label="Decision tree test line")
    plt.plot(DT_x_axis, DT_y_axisTrain, c="g", linestyle="--", label="Decision tree
train line")
    plt.xlabel('# Nodes', fontsize=18)
    plt.ylabel('Accuracy (%)', fontsize=18)
    plt.axis([0, 50, 0.70, 1.01])
    plt.legend(fontsize = 16)
    plt.show()

def Plot_time(X_train,Y_train,X_test):

    # ...This method plot the variation of the training time of each method in front of
their own estimators...

    #Arrays
    RF x axis, RF y axis = [], []
    ET_x_axis, ET_y_axis = [], []
    AD_x_axis, AD_y_axis = [], []
    KNN_x_axis, KNN_y_axis = [], []
    DT x axis, DT y axis = [], []
    new X train, new Y train = [], []

    #Counters
    i = 0
    comp = 0

    # Loop that for each method (comp) compute the maximum Depth varying the estimator
(i)
    for t,n in zip(X_train, Y_train):

        new X train.append(t)
        new Y train.append(n)

        if comp == i:

            #Random Forest
            RF start time = time.time()
            RF clf = RandomForestClassifier(n estimators=51, max depth=16)
            RF_clf.fit(new_X_train, new_Y_train)
            RF_elapsed_time = time.time() - RF_start_time
            RF_x_axis.append(i)
            RF_y_axis.append(RF_elapsed_time)
            print("RF -", i, "---- Time:", RF_elapsed_time)

            #Extra Tree
            ET_start_time = time.time()
            ET_clf = ExtraTreesClassifier(n_estimators=51, max_depth=30)
            ET clf.fit(new X train, new Y train)
            ET elapsed time = time.time() - ET start time
            ET_x_axis.append(i)
            ET_y_axis.append(ET_elapsed_time)
            print("ET -", i, "---- Time:", ET_elapsed_time)

            #KNN
            KNN start time = time.time()
            KNN_clf = KNeighborsClassifier(n_neighbors=30)
            KNN_clf.fit(new_X_train, new_Y_train)
            KNN elapsed time = time.time() - KNN start time
            KNN x axis.append(i)
            KNN y axis.append(KNN elapsed time)
            print("KNN -", i, "---- Time:", KNN_elapsed_time)

            #Adaboost
            AD start time = time.time()
            AD clf = AdaBoostClassifier(n estimators=70)
            AD clf.fit(new X train, new Y train)
            AD elapsed time = time.time() - AD start time
            AD_x_axis.append(i)
            AD_y_axis.append(AD_elapsed_time)
            print("AD -", i, "---- Time:", AD_elapsed_time)
```

```
            #Decision Tree
            DT_start_time = time.time()
            DT_clf = DecisionTreeClassifier(max_depth=8)
            DT_clf.fit(new_X_train, new_Y_train)
            DT_elapsed_time = time.time() - DT_start_time
            DT_x_axis.append(i)
            DT_y_axis.append(DT_elapsed_time)
            print("DT -", i, "---- Time:", DT_elapsed_time)


            print(".................")
            i = i + 1000
        comp = comp + 1

    #Plotting the figure
    plt.rc("xtick", labelsize=18)
    plt.rc("ytick", labelsize=18)
    plt.plot(RF_x_axis, RF_y_axis, c="r", label="Random Forest training model")
    plt.plot(ET_x_axis, ET_y_axis, c="b", label="Extra-Tree training line")
    plt.plot(KNN_x_axis, KNN_y_axis, c="c", label="KNN training line")
    plt.plot(AD_x_axis, AD_y_axis, c="g", label="Adaboost training line")
    plt.plot(DT_x_axis, DT_y_axis, c="y", label="Decision Tree training line")
    plt.xlabel('# stars computed',fontsize=18)
    plt.ylabel('Time (s)',fontsize=18)
    plt.legend(fontsize=16)
    plt.axis([0, 120000, 0, 25])
    plt.show()


def Plot_Importances(data_Sim):

    # ...This method plots the importances level of each astronomic variable in the
training the data...

    #
    df = pd.DataFrame(data_Sim["Data_values"], columns=data_Sim["Feature_names"])
    df['is_train'] = np.random.uniform(0, 1, len(df)) <= .75
    df['species'] = pd.Categorical.from_codes(data_Sim["Type"], data_Sim["Type_names"])


    train, test = df[df['is_train'] == True], df[df['is_train'] == False]
    features = df.columns[0:8]

    forest = RFC(n_jobs=2, n_estimators=55)
    y, _ = pd.factorize(train['species'])
    forest.fit(train[features], y)

    preds = forest.predict(test[features])
    pd.crosstab(index=test['species'], columns=preds, rownames=['actual'],
colnames=['preds'])

    importances = forest.feature_importances
    indices = np.argsort(importances)

    #Plotting the figure
    plt.figure(1)
    plt.rc("xtick", labelsize=20)
    plt.rc("ytick", labelsize=20)
    plt.barh(range(len(indices)), importances[indices], color='b', align='center')
    plt.yticks(range(len(indices)), features[indices])
    plt.xlabel('Relative Importance (%)', fontsize=20)
    plt.ylabel('Physical parameters', fontsize=20)
    plt.xlim(0,0.25)
    plt.show()


#---------------------------------------------------------------------
#####################################################################
#########################  THE MAIN  ################################
#####################################################################
#---------------------------------------------------------------------


def main():
```

```
    # Reading the Simulated sample
    data_Simulated = Read_csv('C:/Users/kanti/Desktop/correctFiles/simulatedTrue.csv')

    # Dictionary of our Simulated sample
    dataset_Sim = {
        "Data_values": data_Simulated.values[:, :-1],
        "Type": data_Simulated.values[:, 8],
        "Type_names": ["Halo", "Thick", "Thin"],
        "Feature_names": ["B", "R", "G", "R.Ascension", "B-R", "Declination", "P.
Motion","H"]
    }

    #Splitting the sample in train and test samples
    X_train, X_test, Y_train, Y_test = train_test_split(dataset_Sim["Data_values"],
dataset_Sim["Type"])


    # -----HERE FOR CALL ALL THESE METHODS FOR MANIPULATING WITH THE SIMULATED SAMPLE---


    #Reading the Observed sample
    data_Observed = Read_csv('C:/Users/kanti/Desktop/correctFiles/observedTrue.csv')


    #Dictionary of our Observed sample
    dataset_Ob = {
        "Data_values": data_Observed.values[:],
        "Feature_names": ["B", "R", "G", "R.Ascension", "B-R", "Declination", "P.
Motion","H"]
    }

    # ---HERE FOR CALL ALL THESE METHODS FOR MANIPULATING WITH THE OBSERVED SAMPLE---



if __name__ == "__main__":
    main()
```